Titre: Title:	Études empiriques des composantes cognitives en processus de développement logiciel
Auteur: Author:	Mihaela Dulipovici
Date:	2005
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Dulipovici, M. (2005). Études empiriques des composantes cognitives en processus de développement logiciel [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7365/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/7365/
Directeurs de recherche: Advisors:	Pierre N. Robillard
Programme: Program:	Unspecified

UNIVERSITÉ DE MONTRÉAL

ÉTUDES EMPIRIQUES DES COMPOSANTES COGNITIVES EN PROCESSUS DE DÉVELOPPEMENT LOGICIEL

MIHAELA DULIPOVICI
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
AVRIL 2005

Your file Votre référence
ISBN: 0-494-01308-7
Our file Notre référence
ISBN: 0-494-01308-7

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

ÉTUDES EMPIRIQUES DES COMPOSANTES COGNITIVES EN PROCESSUS DE DÉVELOPPEMENT LOGICIEL

présenté par : DULIPOVICI Mihaela

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. DESMARAIS Michel, Ph.D., président

M. ROBILLARD Pierre N., Ph.D., membre et directeur de recherche

M. BOUDREAULT Yves, M.Sc., membre

À ma famille, pour sa grande motivation

REMERCIEMENTS

Il y a beaucoup de ressemblances entre un travail de recherche et une expédition dans l'inconnu. Le chercheur doit souvent faire face à des problèmes jamais rencontrés auparavant. Pourtant, il doit trouver en lui seul la force et la motivation d'avancer et d'accomplir son but. La réalisation d'un mémoire de maîtrise est une exploration de ce que nous avons acquis pendant les longues heures d'études et le plus important – une exploration de nous-mêmes.

Dans mon expédition j'ai eu la grande chance de ne pas être seule. Tout au long de mes travaux de recherche j'ai été guidée par les conseils de mon directeur de recherche – le professeur Pierre N. Robillard. Je le remercie pour sa gentillesse, sa disponibilité, et surtout pour ses critiques constructives. Pour la confiance qu'il m'a toujours témoignée, je tiens à lui exprimer toute ma gratitude.

Je remercie également les professeurs Michel Desmarais et Yves Boudreault d'avoir accepté d'être sur le jury de ce mémoire ainsi que pour leurs commentaires constructifs.

Je tiens à remercier très sincèrement Éric Germain pour toute sa collaboration, les discussions que nous eûmes ces années durant, pour avoir partagé son expérience et son temps. Je remercie tous ceux qui ont contribué au déroulement et au succès du cours Atelier de Génie Logiciel. Aussi, ce travail de recherche n'aurait pu être possible sans la collaboration des étudiants qui ont participé au cours Atelier de Génie Logiciel, session Hiver 2002.

La conciliation études – famille est un enjeu majeur. Je voudrais remercier ma famille, particulièrement mon mari, pour l'oreille attentive quand je lui parlais de mon sujet de recherche, pour la patience pendant ce mémoire parfois envahissant, pour les encouragements et le soutien tout au long de mon « retour à l'école ». Un merci spécial à mes deux filles pour la lecture de mes « artefacts écrits », pour leur intérêt pour les processus de développement logiciel et pour énormément d'autres raisons.

RÉSUMÉ

La pression de livrer rapidement des logiciels de qualité, à partir des requis qui changent continuellement, oblige les chercheurs et les praticiens à repenser les pratiques traditionnelles du processus logiciel. D'ailleurs, le développement d'un système logiciel exige une compréhension appropriée non seulement des aspects techniques reliés au processus de développement, mais aussi du facteur humain impliqué dans ce processus. La compréhension des aspects cognitifs entrelacés dans les activités du processus, s'avère d'une importance majeure et, en même temps, un défi pour la recherche.

Le présent mémoire vise l'étude des composantes cognitives en processus de développement logiciel. Le travail de recherche a été entrepris par le biais d'une observation participante réalisée dans le cadre d'une étude de cas portant sur la réalisation en parallèle de projets de développement à partir d'un ensemble unique de requis mais suivant deux modèles de processus distincts. Les résultats de cette recherche sont présentés dans deux articles qui composent le corps de ce mémoire.

Une première étude présente les démarches suivies afin d'identifier un lien entre la qualité des requis et le processus utilisé pour développer un système logiciel. Selon nos résultats, la méthodologie de développement basée sur le triplet « activité – rôle – artefact », qui souligne l'importance des artefacts écrits pour capturer les requis du système, semble plus appropriée pour développer les entités basées sur des requis à caractère dynamique. D'autre part, la méthodologie de développement agile qui capture

les requis du système par communication verbale entre le client et l'équipe de développement semble plus appropriée pour développer des entités basées sur des requis à caractère statique. Les résultats de cette étude peuvent permettre une meilleure compréhension du rôle joué par un processus de développement spécifique et son importance pour la qualité du produit logiciel final.

Une deuxième étude présente une approche originale afin d'identifier et d'analyser les aspects cognitifs spécifiques à la création des artefacts. À cette fin, les termes d'artefact mental et d'artefact physique sont proposés ainsi qu'un modèle cognitif à trois pôles. Bien que les deux processus utilisés soient très différents, les résultats montrent que le même effort est passé sur des artefacts mentaux comme sur des artefacts physiques, dans le contexte des deux processus. Outre cela, l'espace de la structure cognitive à trois pôles permet de tracer la courbe d'évolution pour la création d'artefacts, et ce dans le cas des deux processus utilisés.

Les résultats de nos recherches aident à mieux comprendre les aspects cognitifs du développement logiciel afin de proposer des améliorations aux activités et aux artéfacts qui composent un processus La compréhension du comportement cognitif des développeurs ainsi que du rôle important joué par le processus de développement pourrait être utile aux gestionnaires de projets afin de proposer les actions nécessaires pour améliorer le processus de développement de logiciels

ABSTRACT

The pressure for delivering quality software based on constantly changing requirements compels researchers and practitioners to rethink the traditional software process practices. Moreover the development of a software system requires an appropriate understanding not only of the technical aspects connected with the process, but also of the human factor involved in this process. The understanding of the cognitive aspects, which are interlaced in the activities of the process, is of major importance and at the same time it is a challenge for research.

The present dissertation attempts to address the cognitive components in software development process. The research was carried out by participating in an observational study, based on the parallel realization of two projects with a unique set of requirements, but two different development processes.

A first study presents the steps to identify a link between the quality of the realized requirements and the process used to develop the software system. According to our results, the "activity-role-artefact" based development methodology, which stresses the importance of written artefact for capturing the system requirements, seems more appropriate for developing entities based on dynamic functions. On the other hand, the agile software development methodology, which captures the system requirements through verbal communication between the customer and the development team, seems more appropriate for developing entities based on static functions. The results of this

study may allow better understanding of the role played by a specific process and its importance in the quality of the final software product.

The second study presents an original approach for identifying and analyzing cognitive aspects specific to artefacts creation. For this purpose the notion of mental model was extended to software process and the terms of mental artefact and physical artefact were proposed. Even though the two processes are very different, the results show that, in both cases, the effort spent on mental artefacts is as significant as the effort spent on physical artefacts. A three poles cognitive model is suggested for consideration. The three poles cognitive structure space outlines the evolution curve for artefacts creation in the context of the two processes used.

These findings help researchers and practitioners to better understand the cognitive aspects of the software development in order to suggest improvements in the activities and the artefacts, which compose a process. The understanding of the developers' cognitive behaviour and of the important role played by the development process also helps software project managers in order to designate the actions required for improving the software development process.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES SIGLES ET ABRÉVIATIONS

ASD	Adaptive Software Development
CNRSC	Conseil National de la Recherche Scientifique de Canada
CRC	Class, Responsibility and Collaboration
EQ	Modèle équivalent
EQM	Modèle équivalent, itération de maintenance
EQn	Modèle équivalent, itération n
Gn	Groupe d'artefacts
IEEE	Institute of Electrical and Electronics Engineers
It	Itération
Pd–D–Mg	Predéveloppement – Développement – Gestion
REPLAN	Produit pour la Planification des Réunions
RUP	Rational Unified Process
UML	Unified Modeling Language
Un	Équipe utilisant le processus UPEDU
UPEDU	Unified Process for Education
UP	Unified Process for Education, cycle de développement
UPM	Unified Process for Education, itération de maintenance
UPn	Unified Process for Education, iteration n
XP	Extreme Programming

XP-A	Processus basé sur XP
XPn	Extreme Programming, itération n
XPM	Extreme Programming, itération de maintenance
Xn	Équipe utilisant le processus XP

LISTE DES ANNEXES

INTRODUCTION

Depuis que les gens développent des systèmes logiciels, ils se heurtent à des difficultés. Néanmoins certaines suggestions sont proposées afin d'améliorer les procédures de développement de systèmes logiciels. En réponse aux difficultés rencontrées par les développeurs et les gestionnaires des projets logiciels, Brooks (1987) a présenté les problèmes essentiels concernant le développement de logiciel dans sa célèbre recherche d'une balle d'argent.

De nos jours, l'environnement des affaires a changé considérablement, les systèmes logiciels sont de plus en plus complexes et d'une certaine envergure et, par conséquent, le processus de développement présente de nouvelles dimensions. Une récente définition met en évidence les nouveaux aspects du développement logiciel :

> « Software development is a series of goal-directed, resource-limited, cooperative games of invention and communication. The primary goal of each game is the production and deployment of a software system; the residue of the game is a set of markers to assist the players of the next game. People use markers and props to remind, inspire and inform each other in getting to the next move in the game. The next game is an alteration of the system or the creation of a neighboring system. Each game therefore has as a secondary goal to create an advantageous position for the next game. Since each game is resource-limited, the primary and secondary goals compete for resources.» (Cockburn, 2004, p. 2)

Il est à noter dans la définition allégorique de Cockburn, la complexité du processus de développement logiciel et sa dualité d'objectifs – de réaliser un produit logiciel de qualité et de le faire évoluer, ainsi que l'aspect de coopération et de communication entre les membres de l'équipe de développement. Ces éléments sont traités dans les paragraphes suivants.

Au cœur du développement logiciel se trouvent les individus qui, en utilisant leurs connaissances et leur créativité, essayent de comprendre les problèmes qui changent continuellement, de trouver les meilleures solutions et de les résoudre. Il est souvent impossible d'imaginer la réalisation des systèmes logiciels sans le concours de plusieurs individus réalisant des rôles bien définis dans une équipe de développement. Le « jeu » suggéré par Cockburn (2003, 2004) est un jeu coopératif, où les membres de l'équipe de développement – en tant que joueur – communiquent entre eux et se partagent les informations. Robillard, Kruchten et d'Astous (2002) remarquent la présence d'une démarche exploratoire permettant d'identifier les informations manquantes. Des séquences opportunistes se retrouvent dans toutes les activités sur lesquelles le processus de développement repose. Cette composante opportuniste est de plus en plus observée au fil de l'évolution des projets logiciel.

Le développement de logiciel est un processus complexe impliquant des activités spécifiques multiples qui nécessitent l'acquisition d'un grand volume de connaissance des domaines différents et une variété de processus cognitifs. Il est « a progressive crystallization of knowledge » (Robillard, 1999). L'acquisition et la compréhension des

informations dépendent de plusieurs facteurs, y compris les capacités cognitives des coéquipiers de l'équipe de développement. Les développeurs forment des modèles mentaux pour comprendre le domaine du problème, pour analyser des alternatives différentes de conception ou pour identifier les solutions potentielles (Johnson-Laird, 1983; Norman, 1986). Les divers aspects d'acquisition et de partage de connaissance ont été analysés par des psychologues cognitifs ainsi que par des experts de logiciel, essayant de mieux comprendre le processus cognitif complexe amalgamé dans le développement de logiciel.

Les membres de l'équipe s'échangent leurs connaissances, leurs expériences et leurs idées sur les concepts reliés au logiciel à développer. Dans ce contexte, il est possible, tel que suggéré par Cockburn (2002), de voir la communication entre les coéquipiers comme « touching into a shared experience » (Cockburn 2002). Naur (1992), quant à lui, affirme que ce qui doit être transféré est l'interprétation qui a été construite par chaque membre de l'équipe. Pour un concept donné, chacun a sa propre forme de représentation analogique – son propre modèle mental. Au niveau de l'équipe, après le partage et l'optimisation de modèles mentaux individuels (Levesque, 2001; Yen, 2003), un nouveau modèle mental est créé – le modèle mental de l'équipe (Hinsz, 1990; Klimoski, 1994). Cette idée est également synthétisée par Cockburn « The people who are inventing, manipulating and communicating information must move that information across multiple heads in order to produce the solution » (Cockburn, 2004, p 3).

La communication des informations entre les membres de l'équipe peut prendre plusieurs formes, plus ou moins formelles. Tiré du latin « arte factum », le terme

signifie une partie d'information utilisée ou produite lors du processus de développement d'un logiciel. L'artefact est considéré une voie directe de communication et d'enregistrement des informations, ainsi qu'une forme efficace de transfert de connaissance. L'artefact peut prendre la forme d'un document texte, d'un diagramme, d'un résultat de test ou d'une séquence de code. En outre, avec l'approche « model-driven », les modèles eux-mêmes deviennent des artefacts primaires. Le développement de logiciel est donc une discipline centrée sur les artefacts qui sont utilisés pour capturer les informations spécifiques sur le système logiciel. Les artefacts sont également présents dans la définition du développement logiciel donnée d'une façon allégorique par Cockburn (2004), en utilisant l'analogie « marker and props ». Le rôle de ces derniers est d'informer les membres de l'équipe et de leur inspirer de nouvelles idées. Par ailleurs, les « marker and props » jouent un rôle de base dans le processus d'évolution du système logiciel. La nature évolutive des projets logiciels est très importante; chaque projet logiciel mène vers un autre projet. L'aspect continuel du « jeu » indique son double objectif : le premier objectif est de réaliser un produit logiciel de qualité; le deuxième objectif est de bien préparer l'avenir du projet logiciel et d'assurer sa continuité.

Le premier objectif du processus logiciel qui est de produire un logiciel de qualité est plus qu'un des principaux sujets d'intérêts, il est un défi pour les développeurs de logiciel. La qualité est un concept élusif. Kitchenham (1996) affirme que la qualité est « hard to define, impossible to measure, easy to recognise ». D'ailleurs la qualité est une notion composée qui repose sur plusieurs perspectives. Usrey et Dooley (1996)

synthétise la perspective de l'utilisateur du système logiciel : « quality is in the eye of the beholder ». Cette définition de la qualité fait particulièrement référence au degré auquel un système logiciel atteint les besoins de l'utilisateur. Dans cette perspective, tous les aspects reliés aux fonctionnalités du système logiciel sont de la plus grande importance pour l'utilisateur. Une tâche fondamentale dans le développement des systèmes de logiciel est de communiquer les requis du système aux développeurs, tâche qui est accomplie par la création des artefacts. De plus en plus, le processus est perçu comme un élément important pour améliorer la qualité du produit logiciel.

D'ailleurs, le processus de développement logiciel pourrait être vu comme un artefact complexe, qui n'est pas très différent d'un logiciel complexe et distribué. L'idée a été synthétisée par Osterweil (1987) : « Software processes are software, too ». Un processus de développement logiciel est après tout, le programme exécuté par les membres de l'équipe de développement, qui utilisent leurs connaissances et leurs capacités cognitives, qui partagent leurs solutions pour obtenir à la fin un système logiciel de qualité. Les praticiens et les chercheurs en génie logiciel travaillent, depuis plus d'une trentaine d'années, à édifier des processus permettant le développement efficace et efficient des logiciels de qualité. Un processus logiciel est la définition des relations qui existent entre les activités, les rôles et les artefacts nécessaires à la réalisation d'un projet logiciel (Robillard, Kruchten et d'Astous, 2002). Cependant, il n'y a pas un consensus sur les principes du processus ou sur la manière que les développeurs doivent utiliser pour communiquer des informations pendant le développement de logiciel. Deux philosophies principales de développement de logiciel

apparaissent. La méthodologie de développement logiciel basée sur le triplet « activité – rôle – artefact » est une approche systématique pour les activités du processus et souligne l'importance des artefacts écrits (Kruchten, 2000). En réaction à ce type de méthodologie il y a, depuis quelques années, un intérêt croissant pour les méthodologies dites de type agile, qui font de la communication verbale une des valeurs principales (Manifesto for Agile Software Development – Agile Alliance, 2004). Ces deux méthodologies recommandent des voies différentes pour communiquer des informations.

Il est intéressant de noter que malgré le débat sur les deux philosophies, les résultats présentés ramènent à des études sur une philosophie ou sur une autre et rarement à des études comparatives. La principale difficulté pour comparer les deux approches provient du fait que l'étude comparative doit se faire dans un contexte permettant de rendre valide la comparaison. D'ailleurs, le développement d'un système logiciel exige non seulement une compréhension appropriée de l'espace du problème et des aspects techniques reliés au processus, mais aussi du facteur humain impliqué dans ce processus de développement.

Vu les aspects mentionnés, nous pouvons faire la remarque qu'une tâche fondamentale dans le développement des systèmes de logiciel est de communiquer l'information à l'interne ainsi qu'à l'externe de l'équipe de développement. Une solution d'accomplir cette tâche est par la création des artefacts. En outre, le rôle des artefacts augmente surtout dans le cas du deuxième objectif du processus de développement logiciel identifié par Cockburn (2004) – préparer en vue de l'évolution de logiciel.

À la lumière de ces constats, ce mémoire vise les composantes cognitives en processus de développement logiciel. Tel que mentionné par Cockburn dans sa définition allégorique du développement logiciel, des aspects spécifiques aux deux objectifs du processus logiciel seront abordés, et ce à travers deux questions de recherche.

Bien qu'il soit reconnu qu'un choix approprié du processus de développement peut améliorer la qualité du produit, il y a peu d'évidence empirique sur les liaisons directes entre le processus et la qualité de réalisation des requis du produit logiciel. Dans ce sens, la première question de recherche vise à déterminer l'influence possible du processus de développement sur la qualité de réalisation des requis. Ainsi, les mêmes requis ont servi à deux groupes de développeurs logiciels : un groupe qui a utilisé le processus structuré UPEDU (Unified Process for Education) et un deuxième groupe qui a utilisé un processus de type agile – XP (Extreme Programming).

La deuxième question de recherche vise à examiner le comportement des développeurs lors de la résolution de problèmes tout au long d'un processus logiciel. L'analyse est basée sur l'effort de réalisation des artefacts logiciel significatifs, pour deux processus différents : UPEDU et XP. L'étude observatoire est basée sur une approche originale permettant à la fois d'identifier et d'analyser les aspects cognitifs propres à la création des artefacts.

Bien que certains chercheurs aient souligné l'importance du sujet, aucune recherche connue à ce jour n'a tenté de présenter des trouvailles concernant ces deux questions. Les résultats de nos recherches aideront à mieux comprendre l'aspect cognitif du

développement logiciel afin de proposer des améliorations aux activités et aux artéfacts qui composent un processus. Nous croyons que les résultats obtenus par cette recherche peuvent permettre une meilleure compréhension du rôle joué par un processus de développement spécifique et contribuer à faire progresser l'état des connaissances dans le domaine du développement logiciel.

CHAPITRE 1

REVUE CRITIQUE DE LA LITTÉRATURE

Une supposition largement reconnue dans le développement de logiciels est « si vous avez une méthode, vous savez comment développer le logiciel avec succès ». Différentes méthodes existent actuellement (Lindvall et Rus, 2000; Fuggetta, 2000), couvrant plusieurs perspectives du développement de systèmes logiciel. La pression de livrer rapidement des logiciels, respectant les critères de qualité, mais à partir des requis qui changent continuellement obligent les chercheurs et les praticiens à repenser les pratiques traditionnelles du processus logiciel. Cependant, le domaine n'a pas atteint de consensus à ce qui constitue une bonne méthode de développement.

D'ailleurs, il est de plus en plus supporté dans la littérature que multiples problèmes rencontrés au cours du développement de logiciels peuvent aussi être attribuables à un autre facteur, soit le facteur humain du processus logiciel. Dans la communauté de génie logiciel, le comportement cognitif des individus qui développent le logiciel devient un sujet d'un grand intérêt, jadis propre aux sciences psychologiques et sociologiques.

Ce chapitre vise à faire une revue critique de la littérature traitant des aspects liés au développement logiciel, soit les composantes cognitives en processus logiciel.

1.1. Processus de développement logiciel

Les processus de développement logiciel visent à énoncer les meilleures pratiques acquises par les experts ayant réussi à produire des logiciels de qualité tout en respectant les contraintes de temps et de coût (Cugola et Ghezzi, 1998). À cet égard, le processus Rational Unified Process (RUP) est le plus complexe et constitue un standard. Toutefois, un très sérieux concurrent se taille une place dans ce monde perpétuellement en mouvement. Il s'agit de la famille des méthodes agiles. Depuis quelques années, le débat sur le meilleur processus continue à être un sujet retrouvé dans un grand nombre de revues de spécialité. « Today, a new debate rages: agile software development versus rigorous software development. » (Highsmith, 2002) – une affirmation qui est toujours d'actualité. Les sections suivantes présentent les méthodologies les plus représentatives des deux approches, soit la méthodologie *Rational Unified Process* – représentant de l'approche traditionnelle de développement de logiciels et *Extreme Programming* – représentant les méthodologies agiles.

1.1.1. Le processus « Rational Unified Process »

Le *Rational Unified Process* (RUP) tel que mentionné par son auteur (Krutchen, 2000) est un « processus d'ingénierie de logiciel ». Premièrement, il fournit une approche disciplinée à l'assignation de tâches et des responsabilités pour le développement de logiciel. Deuxièmement, RUP est un « produit processus » qui regroupe dans un site web personnalisable l'ensemble d'activités pour transformer les requis des utilisateurs dans un système logiciel. Troisièmement, RUP est également un

« processus cadre » qui permet de s'adapter aux besoins ainsi qu'aux caractéristiques spécifiques des compagnies de développement logiciel.

De plus, RUP est un processus à deux dimensions :

- La première dimension représente l'aspect dynamique du processus et montre les aspects du cycle de vie du processus.
- La deuxième dimension représente l'aspect statique du processus et montre les disciplines fondamentales qui regroupent logiquement les activités du processus.

Le processus itératif et incrémental, tel que définit par RUP, est guidé par les cas d'utilisation et centré sur l'architecture. Il a été développé conjointement avec UML, qui fait partie intégrante du processus. Le modèle conceptuel fondamental de RUP se base sur le triplet « activité – rôle – artefact » (voir Figure 1.1). Un processus logiciel est donc constitué par les relations qui existent entre les activités, les rôles et les artefacts nécessaires à la réalisation d'un projet logiciel (Robillard, Kruchten at d'Astous, 2002).

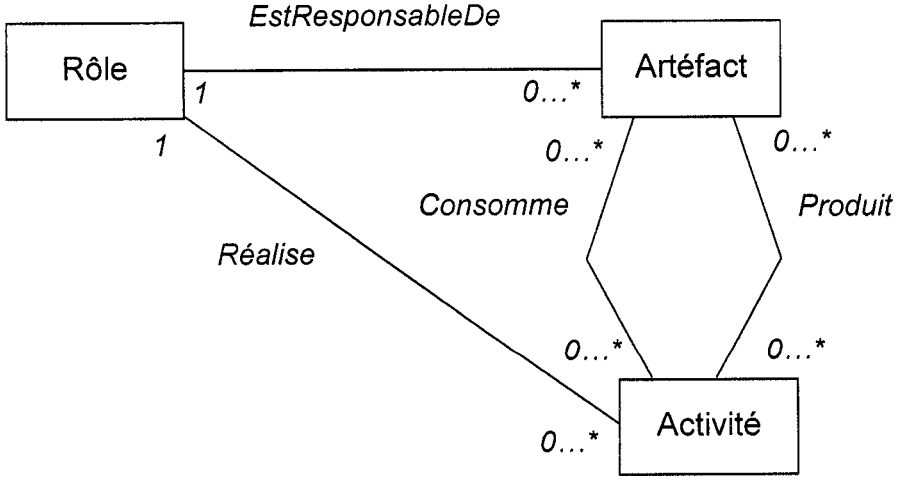

Figure 1.1 Modèle conceptuel fondamental de RUP
(source : Robillard, Kruchten at d'Astous, 2002)

Un rôle définit les responsabilités des individus qui effectuent des activités durant le processus. Un rôle peut réaliser plusieurs activités. Une activité est la plus petite pièce de travail effectuée durant le processus. Chacune des activités peut être réalisée par un seul rôle. Une activité consomme un certain nombre d'artefacts à l'entrée et produit également un certain nombre d'artefacts à la sortie. Un artefact est une pièce d'information produite, modifiée ou utilisée par une activité durant le processus. Un artéfact peut être produit par plusieurs activités.

La méthodologie basée sur le triplet « activité – rôle – artefact » souligne l'importance des artefacts écrits pour capturer les requis, pour présenter les décisions de conception d'architecture, pour concevoir les cas de test ou pour faire la planification du projet. Les artefacts incluent des documents (environ la moitié des artefacts RUP est classifiée comme documents), des diagrammes, de modèles, des rapports, des cas de test ou des séquences de code. Il est à remarquer que les documents peuvent être constitués du texte et des diagrammes. D'ailleurs, RUP décrit plusieurs artefacts qui sont composés d'autres artefacts. Au total RUP propose plus de cent artefacts, pour pouvoir répondre aux différents besoins des projets. Malgré ce grand nombre, les artefacts sont importants pour l'information qu'ils contiennent. Une tâche fondamentale dans le développement de systèmes logiciel est de communiquer l'information. Dans le cas des méthodologies de développement basées sur le triplet « activité – rôle – artefact », cette tâche est accomplie par des artefacts écrits. Ils représentent le medium de communication de l'information spécifique au projet logiciel.

1.1.2. Le processus « Extreme Programming »

En réaction à ce type de méthodologie et particulièrement à la grande quantité de documentation écrite requise, il y a, depuis quelques années, un intérêt croissant pour les méthodologies dites de type agiles (Manifesto for Agile Software Development – Agile Alliance, 2004). Ces méthodologies plus souples visent à faciliter l'intégration des changements qui peuvent survenir pendant toutes les étapes du processus de développement. Quelques exemples de méthodologies de type agile sont XP (Extreme Programming) (Beck, 1999), ASD (Adaptive Software Development) (Highsmith, 2000), Scrum (Rising et Janoff, 2000), Crystal (Cockburn, 2002; Highsmith, 2002).

La compagnie Cutter Consortium a présenté les résultats d'une enquête menée par Standish Group (2001) sur l'utilisation des méthodologies de développement de logiciel. 200 compagnies de partout dans le monde et oeuvrant dans différents domaines ont répondu à l'enquête. Les données sont représentatives et elles semblent bien exprimer la réalite (voir la Figure 1.2). Parmi toutes ces méthodologies, l'Extreme Programming (Beck, 1999, 2000) est celle qui a reçu la plus grande attention.

L'Extreme Programming est basée sur un ensemble de pratiques qui régissent les activités d'un projet logiciel d'une manière assez stricte. Pourtant, les auteurs d'Extreme Programming préfèrent parler de système de pratiques ou encore de discipline de développement, plutôt que de méthode.

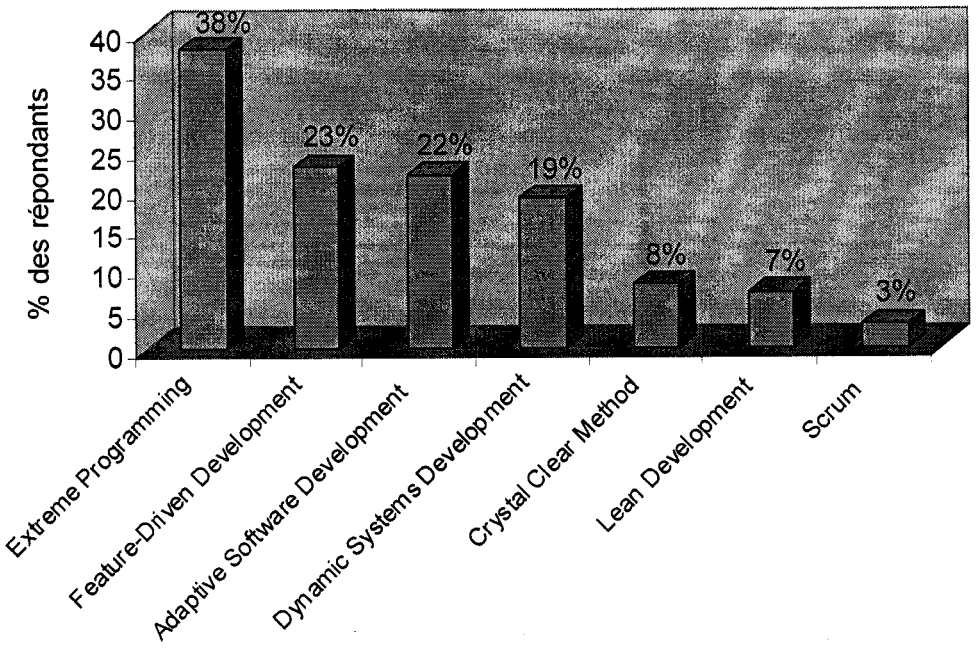

Figure 1.2 L'utilisation des méthodologies agiles
(source : Charrette, 2001)

Avec les pratiques, les rôles font véritablement partie de la culture XP. Quant à la correspondance des rôles XP avec des rôles traditionnels, elle n'est pas toujours immédiate. Parmi ces rôles, nous retrouvons le client qui fait partie de l'équipe de développement et qui est responsable de définir les requis initiaux pour le système à développer. L'analyse de ces requis est réalisée au cours des conversations entre le client-utilisateur et les autres membres de l'équipe de développement. Pour mieux comprendre le système à développer, ils utilisent des métaphores, partagées entre le client et les programmeurs. La méthodologie de type agile fait de la communication verbale une de ses valeurs principales et propose une approche minimaliste à la documentation écrite (Jeffries, Anderson, Hendrickson, 2001).

Quant aux activités, il semble que trois activités soient suffisantes afin de réaliser un logiciel : coder, tester, écouter... Un aspect important est sans aucun doute l'emphase des tests sur le code et l'intégration continue du code. La méthode XP met réellement l'accent sur les tests qui débutent dès le premier jour d'implémentation. Ainsi, dès le départ les tests sont écrits, de façon à rendre le codage plus facile et plus pertinent. Un autre aspect important est le « refactoring » – activité qui signifie d'éliminer la redondance, d'enlever les fonctionnalités inutilisées et de rajeunir la conception. Le « refactoring » a pour effet de garder le code clair, simple, concis et plus facile à comprendre. XP ignore la pratique de type « BigDesignUpFront » et favorise la conception simple, la métaphore, la collaboration d'utilisateurs et des programmeurs, et surtout la communication verbale. L'activité clé de la méthode XP est le codage.

En considérant l'approche XP, il n'est peut-être pas surprenant que dans les livres et les articles que nous avons consultés, les termes « artefact » et « produit de travail » n'apparaissent pas dans les index. Cependant, tel que Smith le remarque « ... it's not difficult to read through the text and pick out references that are artefacts » (Smith, 2001). En mettant sous la loupe les recommandations d'XP, Smith identifie une trentaine d'artefacts, dont certains sont des artefacts composés. D'ailleurs, les praticiens XP recommandent la réalisation écrite de ces artefacts seulement si nécessaire. Toute l'information reliée au développement du logiciel se véhicule par voie orale. Encore une fois, XP fait la preuve d'être une méthodologie axée sur les principes de communication verbale et de simplicité. Le tableau ci-dessous fait une correspondance entre les artefacts de RUP et de XP.

Tableau 1.1 Correspondance entre les artefacts RUP et XP

XP	RUP
Histoires Documentation complémentaire de conversations client – développeur	Vision Glossaire Modèle de cas d'utilisation
Contraintes	Spécifications Supplémentaires
Test d'acceptation Test unitaire Données de test Résultats de test	Modèle de test
Code du logiciel	Modèle d'implémentation
Release	Produit Note de Release
Métaphore	Document d'architecture du logiciel
Conception – CRC, UML, croquis Tâches Documents de conception produits à la fin du projet Documentation de support	Modèle de conception
Standards de codage	Directives de conception Standards de programmation
Zone de travail (développement et d'autres facilités) Outils-cadre de test	Outils

Plan de livraison Histoires estimées Tâches estimées Plan d'itération	Plan de développement de logiciel Plan d'itération
Plan complet du budget	Cas d'affaire Liste de risques Plan d'acceptation de produit
Rapport sur progrès Enregistrement du temps de travail des tâches Données de métriques : Ressources, Portée, Qualité, Temps Autre métriques Trace de résultats Rapports et notes sur de réunions	Évaluation de la qualité du produit Métriques Plan de mesure
Défauts et données associées	Demandes de changement
Outils de gestion de code	Plan de gestion de configuration Repositoire de projet Espace de travail
Idée de pointe (spike)	Prototypes
	Plan de développement Cadre spécifique du projet

Une fois de plus, les artefacts sont des médias de communication ainsi qu'une forme efficace de transfert de connaissance. La documentation est « one of the most important ways to improve program comprehension » (Sametinger, 1995). En outre, du point de vue du client, « documentation is arguably more important than programming » (Covington, 1985). Cockburn (2002) décrit divers modes de communication que les développeurs utilisent en travaillant ensemble (voir Figure 1.3).

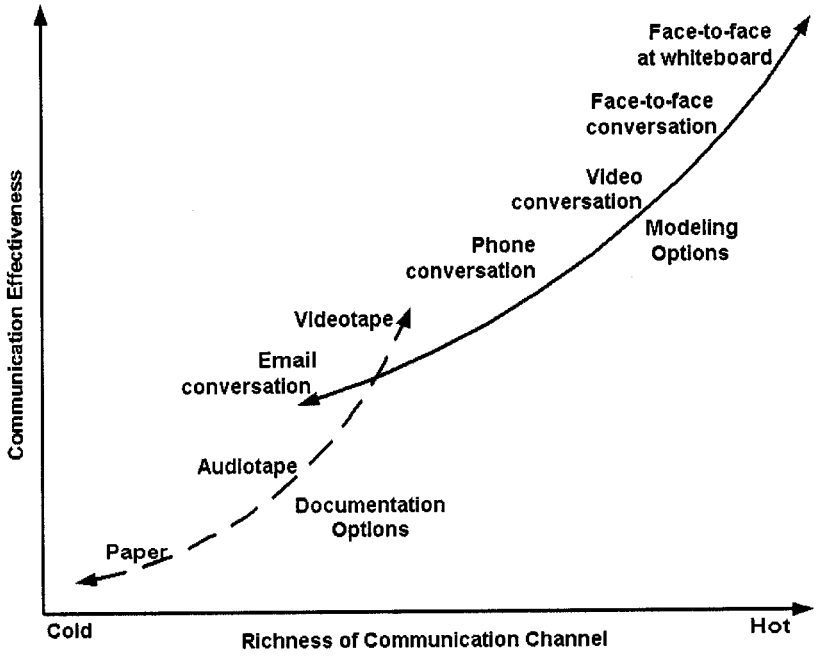

Figure 1.3 Les canaux de communication
(source : Cockburn, 2002)

Les options pour les médias de communication qui pourraient être disponibles sur papier se retrouvent à l'extrême gauche de l'arc en pointillé. Quant à l'autre courbe, une progression vers la gauche implique une perte de communication multimodale ainsi que la possibilité d'interroger – répondre en temps réel. Ceci est bien le cas pour des

processus traditionnels (le RUP en est un exemple). Cockburn (2002) soutient l'idée que la communication la plus efficace est de type personne-à-personne, face-à-face – l'extrême droit de l'arc. C'est le cas pour les processus agiles, en particulier pour XP. En effet, pour une communication efficace, l'information doit être partagée et ce partage doit être bidirectionnel. En utilisant un processus basé sur le triplet « activité – rôle – artefact » les développeurs utilisent un type de « cold communication media » pour communiquer les informations nécessaires. En utilisant une méthode agile, en particulier XP, les développeurs utilisent un type de « hot communication media » pour communiquer les informations nécessaires tant à l'intérieur qu'à l'extérieur de l'équipe.

Parmi les aspects qui différencient les deux approches de développement logiciel, le choix du media de communication qui fait implicitement appel aux artefacts est un des aspects les plus importants. L'analyse du type de transmission d'information entre les membres d'une équipe de développement implique l'analyse des aspects cognitifs en processus logiciel. Ces aspects sont développés dans le sous-chapitre suivant.

1.2. Aspects cognitifs en processus logiciel

Au cœur du développement logiciel se trouvent les gens qui, en utilisant leurs connaissances et leur créativité essayent de comprendre des problèmes qui changent continuellement, de trouver des solutions et de les résoudre. La communication aide à établir une passerelle entre l'expression des requis logiciel et les solutions potentielles. Brooks (1987) suggère que l'essence du développement de systèmes soit un processus créateur. Le développement des systèmes logiciel demande aux développeurs de comprendre les besoins des utilisateurs et de les translater dans un modèle qui peut être implémenté comme un système logiciel. Le développement de systèmes est aussi un ensemble d'activités cognitives que les développeurs exécutent. Un modèle cognitif décrit le processus cognitif et les structures d'informations utilisées pour former un modèle mental. (Storey, Fracchia et Muller, 1997). Les développeurs érigent des représentations mentales, pour mieux comprendre le problème à résoudre, pour analyser et évaluer les alternatives différentes de conception ou pour identifier les solutions potentielles. Par conséquent, nous pouvons déduire que le processus de développement logiciel dépend de la capacité des développeurs de formuler rapidement et correctement un modèle mental et ensuite de simuler et prédire des résultats probables par rapport à ce modèle mental.

La littérature sur les aspects cognitifs en génie logiciel est clairsemée. Le terme de modèle mental est introduit par Johnson-Laird (1983) qui développe une théorie cognitive pour expliquer le raisonnement déductif humain, en général. La même année,

Norman (1983) introduit la distinction entre le modèle mental de l'utilisateur et celui du concepteur. Norman donne également une définition très explicite du terme :

> « In interacting with the environment, with others, and with the artifacts of technology, people form internal, mental models of themselves and of the things with which they are interacting. The models provide predictive and explanatory power for understanding the interaction. » (Norman, 1983, p.7).

Young (1983) pousse plus loin l'analyse de cette distinction, en affirmant qu'il y a plusieurs types différents de représentations crées par les utilisateurs, ainsi qu'en définissant le terme « modèle conceptuel de l'utilisateur ». Norman (1986) révise ces termes et présente une théorie plus raffinée des modèles divers. Le modèle mental est un concept utilisé souvent dans la littérature sur l'interaction homme-machine (Sasse, 1992). La figure 1.4 donne un exemple des interactions des modèles mentaux.

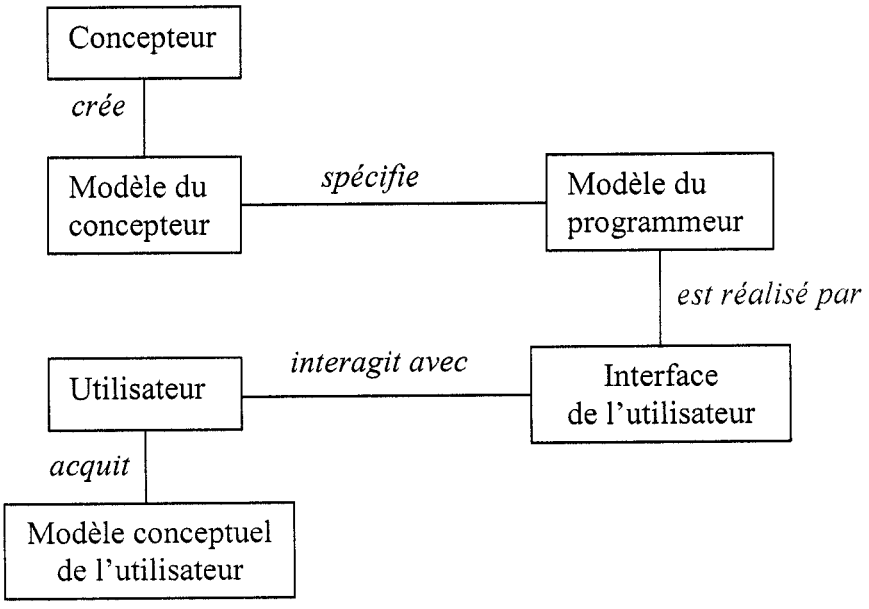

Figure 1.4 L'interaction du modèle de l'utilisateur et du modèle de concepteur (source : Hudson, 2003)

La théorie constructiviste prétend que la connaissance n'est pas transmise, mais plutôt encapsulée dans des structures cognitives. Chaque individu construit son propre modèle mental (von Glasersfeld, 1995). Le développement logiciel est un domaine intensivement basé sur les connaissances des développeurs et qui se fait principalement dans un travail collaboratif d'équipe (Robillard, 1999). En conséquence, nous pourrions considérer l'application de la théorie des modèles mentaux au niveau du processus logiciel. Cette idée est soutenue par Chen (1992) « ... mental models are important not only to the use of software but to the development of software as well ».

Les modèles mentaux sont une forme de représentation analogique des connaissances et des pratiques. Outre cela, il existe une correspondance directe entre les entités et les relations présentes dans la structure de représentation et entre les entités et les relations de ce qu'on cherche à représenter. Un modèle mental a donc une certaine structure, faisant les associations entre les éléments divers de connaissance. De la même manière, les modèles mentaux peuvent être définis comme un schéma ou une représentation spécifique d'un objet ou d'un système. Cependant, il n'existe pas un seul modèle mental « exact » qui correspond à un objet ou à un système (Payne, 1991, 1992). À chaque objet peuvent s'accorder plusieurs modèles à la fois, même s'il est supposé qu'un seul de ceux-ci s'accorde d'une façon optimale. Une définition qui englobe ces aspects est donnée par Doyle :

> « A mental model ... is a relatively enduring and accessible, but limited, internal conceptual representation of an external system (historical, existing,

or projected) whose structure is analogous to the perceived structure of that system. » (Doyle,1999, p.5)

Par l'intermédiaire des modèles mentaux, le développeur peut comprendre, expliquer, déduire et prévoir les résultats pour prendre une décision ou pour entreprendre une action. (Johnson-Laird 1983; Bliss et Ogborn, 1989; Renk, Branch et Chang, 1993).

De plus, n'importe quel modèle mental peut être « exécuté ». Le résultat de l'exécution interne de ce modèle mental est extériorisé. Par la suite, des nouvelles informations s'ajoutent et un nouveau modèle prend forme et ainsi le cycle recommence. Ce cycle est basé sur la théorie de réglage des actions présentée par Hacker (1994) et référée également par Rauterberg (1999) (voir Figure 1.5)

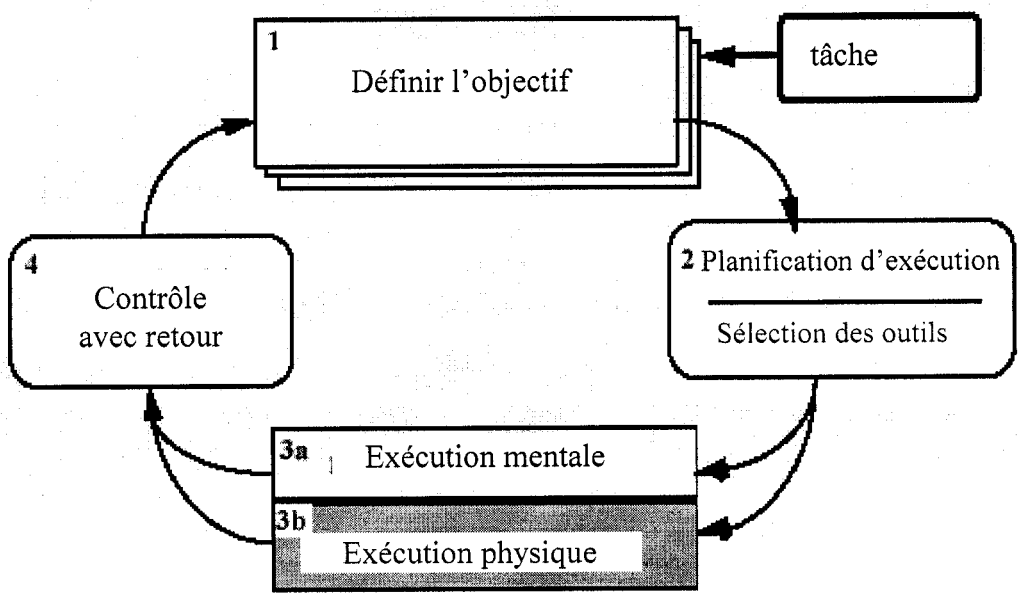

Figure 1.5 Cycle de résolution de problème à l'aide d'un modèle mental (source : Rauterberg, 1999)

Ce concept forme la base de la théorie socioconstructiviste, initialement développée par Vygotsky (1978). La connaissance humaine est affectée par les interactions avec d'autres connaissances et par conséquent les modèles mentaux individuels sont influencés par l'interaction sociale des individus. Évidement, plusieurs stimuli causent la création des modèles mentaux (Newman, 2003). Parmi ces stimuli nous retrouvons l'expérience pratique propre à l'individu, la formation, la lecture de la documentation ou l'observation du travail des coéquipiers. La Figure 1.6 montre certains stimuli qui contribuent à la formation des modèles mentaux.

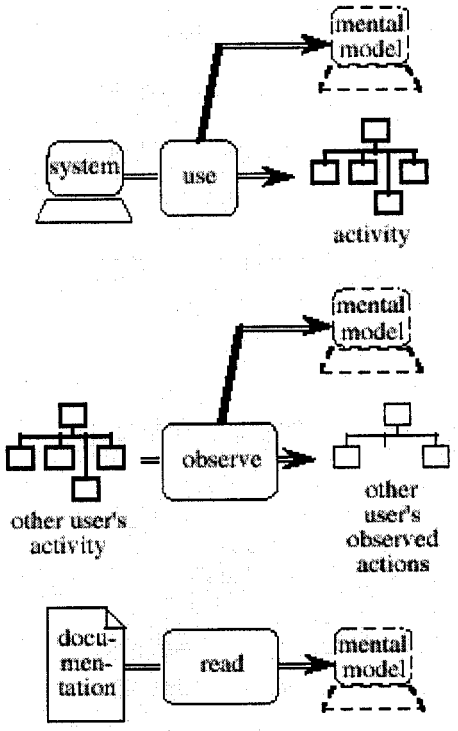

Figure 1.6 Exemple de stimulus pour la création des modèles mentaux (source : Newman, 2003, p.240)

La plupart des projets de logiciel sont le résultat d'un effort d'équipe. Le partage d'information entre les membres de l'équipe de développement nous mène vers l'idée

d'un modèle mental distribué. Les modèles mentaux distribués sont des constructions hypothétiques qui étendent la notion de modèle mental individuel à un contexte d'équipe (Levesque Wilson et Wholey, 2001; Yen, Fan, Wang, Chen et Kamali, 2003). Pour un même sujet, chaque membre d'équipe a son propre modèle mental, qui est plus ou moins différent de celui de son collègue d'équipe. Une communication efficace entre les coéquipiers permet la construction d'un nouveau modèle mental basé sur les modèles individuels. Les développeurs doivent dialoguer à plusieurs reprises afin d'expliquer leurs réflexions et de comprendre les idées des autres (Mathianssen, 2002). La Figure 1.7 montre la manière de composition d'un modèle mental distribué.

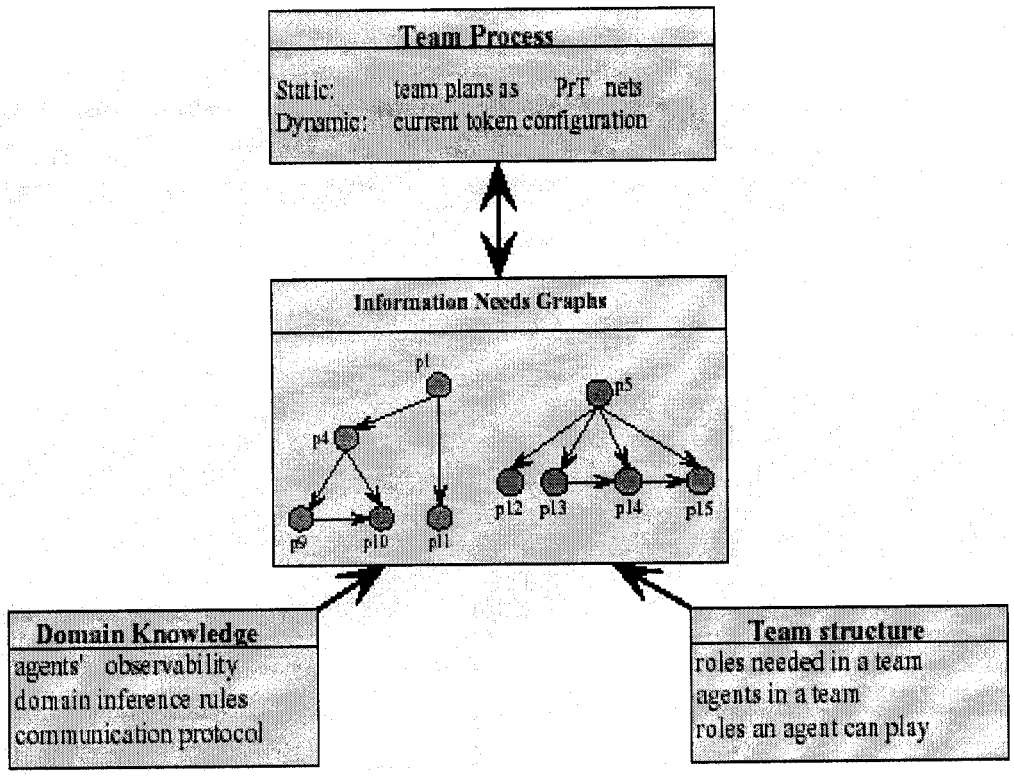

Figure 1.7. La composition d'un modèle mental distribué
(source : Yen, 2003, p5)

A cet égard, un exemple spécifique au domaine du développement logiciel est donné par Holt qui soutient que « software architecture is most usefully thought of as a shared mental model » (2002). Son point de vue est soutenu par le principal rôle de l'architecture logiciel, soit de faciliter la compréhension ainsi que la communication à l'interne et à l'externe de l'équipe de développement.

De même, Klimosky et Mohammed (1994) mentionne plusieurs chercheurs portant attention à la création des modèles mentaux d'équipe. Le sujet est d'un grand intérêt pour les chercheurs qui analysent les performances des équipes (Mohammed et Dumville, 2001). Des études présentent des concepts qui situent les activités cognitives au niveau du travail d'équipe et l'existence même d'une « mémoire d'équipe » (Klimosky et Mohammed, 1994).

Malgré l'importance prouvée des modèles mentaux individuels et d'équipe et malgré l'intérêt des chercheurs des divers domaines à ce sujet, les modèles mentaux ont aussi des limites (Hill et Levenhagen, 1995).

D'abord, les modèles mentaux peuvent ne pas être stables au long du temps. Un modèle qui est approprié à un certain environnement aujourd'hui, peut ne pas être adéquat pour l'environnement de demain. De plus, la nature parcimonieuse des modèles mentaux limite aussi leur précision. Plus un modèle est détaillé – plus son application est précise. Cependant, les détails supplémentaires peuvent diminuer sa valeur en lui ajoutant des éléments ambigus. Les métaphores sont un très bon exemple puisque par leur nature incomplète, elles ajoutent autant de forces que de faiblesses aux représentations mentales. Finalement, les modèles mentaux ne sont pas directement

observables. Leur mesure continue à être un défi, étant donné le fait que les modèles mentaux sont dynamiques et prennent souvent des formes multiples.

Un autre aspect très important est le fait que le produit logiciel ne soit pas semblable aux autres produits de génie. Armour (2004) mentionne les aspects particuliers des produits logiciels et affirme que « the true product is not the software, it is the knowledge contained in the software ». En se basant sur cette affirmation, le processus de développement logiciel devient un regroupement d'activités d'acquisition de connaissance.

La connaissance est une notion élusive basée sur le paradoxe qu'elle réside dans l'esprit de quelqu'un et en même temps elle peut être capturée, représentée et enregistrée. Deux types de connaissance peuvent être identifiés : la connaissance explicite (issues réelles – artéfacts) et la connaissance implicite ou la connaissance tacite. Les modèles mentaux représentent la connaissance tacite. La connaissance est crée par les interactions entre la représentation explicite de la connaissance et la représentation tacite de la connaissance. Quand la connaissance tacite est rendue explicite, la connaissance est cristallisée. Ce processus se produit en spirale qui passe par tacite et explicite, à plusieurs niveaux (Nonaka, Toyama et Konno, 2000). La Figure 1.8 montre la manière de création de connaissance en suivant ce processus en spirale.

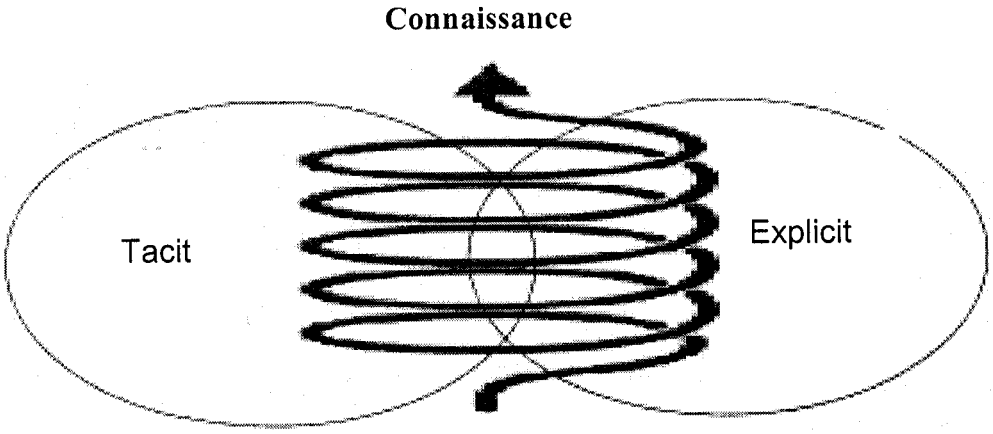

Figure 1.8 La cristallisation de connaissance
(adaptée de Nonaka, Toyama et Konno, 2000, p6)

Les modèles mentaux représentant la connaissance tacite jouent un rôle important en améliorant la résolution des problèmes, tel qu'il est le cas du développement de logiciel.

1.3. Conclusion

La revue de littérature nous a permis d'observer que le développement logiciel est exprimé à l'aide de trois éléments de base : le produit, le processus et les individus. Outre cela, le développement de systèmes logiciel est une activité complexe et surtout une activité cognitive. Au coeur du processus de développement se retrouvent les structures cognitives des développeurs.

Les concepts généraux entourant nos questions de recherche sont présentés dans les premières sections de ce chapitre. Nos deux questions de recherche reposent sur ces concepts généraux, mais demandent également l'exploration des aspects particuliers, spécifiques à chaque question. Ces aspects spécifiques et leurs fondements scientifiques

retrouvés dans la littérature sont présentés dans le cadre de deux articles qui répondent à nos deux questions de recherche. Le chapitre suivant présente l'approche de recherche de ce travail ainsi que la structure générale du mémoire.

CHAPITRE 2

ORGANISATION GÉNÉRALE DU TRAVAIL

2.1. Objectif du mémoire

L'ingénierie logiciel est une industrie omniprésente où la complexité des produits et la vitesse d'apparition de nouvelles technologies augmentent continuellement. Les systèmes logiciels sont développés par des équipes de professionnels qui mettent en jeux leurs connaissances, leur expérience et leur créativité afin de réaliser un produit logiciel de meilleure qualité. Depuis quelque temps, le processus de développement est considéré comme un élément déterminant pour améliorer la qualité du produit logiciel. A l'heure actuelle plusieurs processus existent, mais il n'y a pas un qui est universellement reconnu. Comment déterminons-nous qu'est-ce qui nous aide à choisir le processus le plus approprié pour un certain projet ? Quel processus choisir afin d'obtenir un produit de qualité ? Pour répondre à ces questions, nous devons aller jusqu'au fond de la controverse au sujet des processus et découvrir les dimensions sur lesquelles les processus varient.

Par ailleurs, la structure cognitive du développeur de systèmes est parmi les caractéristiques les plus critiques dans le développement de logiciel. De ce fait, il y a un besoin constant de mieux comprendre les aspects cognitifs impliqués dans le processus logiciel et de développer de nouveaux modèles qui faciliteront cette compréhension.

Ce mémoire vise donc à présenter les dimensions sur lesquelles varient les deux processus de développement logiciel les plus controversé à date et d'analyser les aspects cognitifs impliqués dans ces dimensions (Stacy, Macmillan, 1995).

Pour ce faire, deux questions de recherche ont été développées. La première question vise à déterminer l'influence possible du processus de développement sur la qualité du produit final, en particulier sur la qualité de réalisation des requis. La deuxième question vise à examiner plus en détail le volet cognitif lors de la réalisation des artefacts logiciel significatifs aux deux processus de développement.

2.2. Définition du cadre de recherche

En raison, entre autres, de l'intérêt croissant des aspects humains du génie logiciel, la recherche empirique jadis propre aux domaines reliés aux sciences humaines est de plus en plus utilisée en génie logiciel. « Empirical studies are crucial to the evaluation of processes and human-based activities » (Wohlin et al, 2000). En se basant sur cette affirmation, nous pouvons prétendre que notre approche de recherche empirique a été justifiée puisque dans le processus de développement logiciel l'homme joue un rôle central.

La présente recherche a été inspirée par plusieurs études précédentes entreprises dans le contexte du cours projet « Atelier Génie Logiciel » à l'École Polytechnique de Montréal.

Les étudiants inscrits dans ce cours développent un système logiciel fonctionnel, basé sur une spécification de requis fournie. Toutes les équipes d'étudiants sont en

compétition sur le même projet, et développent leur propre version du système basée sur une spécification des requis unique. Le développement du logiciel doit se faire tout en respectant un processus logiciel bien précis. Le processus utilisé dans le cadre de l'Atelier est le UPEDU (Unified Process for Education) (Robillard, Kruchten, et d'Astous, 2001, 2002) qui est une version allégée du RUP (Rational Unified Process) (Kruchten, 2000) adaptée à un environnement académique. La structure de ce type de cours-projet est présentée en détail par Robillard (1996). Dans le contexte de l'Atelier des recherches ont été effectuées dans le but d'identifier les problèmes qui doivent être résolus afin d'établir des pratiques efficaces. Les résultats de plusieurs études sur la répartition de l'effort de développement logiciel, des activités exécutées et la qualité d'artefacts font l'objet de quatre articles de conférence auxquels l'auteur a contribué.

L'Atelier Génie Logiciel est un cours s'échelonnant sur un semestre pour des étudiants de quatrième année et, en même temps, une plate-forme de recherche pour les aspects spécifiques du génie logiciel. Ce mémoire présente les résultats d'un travail de recherche approfondi qui a porté sur l'édition 2002 de l'Atelier.

2.2.1. Le processus

Le contexte de l'Atelier 2002 est unique par utilisation de deux processus différents : UPEDU (Unified Process for Education) et un processus basé sur XP (Extreme Programming).

UPEDU est un processus logiciel complet qui définit toutes les activités et les artefacts nécessaires afin de réaliser le projet dans le cadre de l'Atelier. Les

caractéristiques de RUP, tels que mentionnées dans le chapitre précédent s'applique également au UPEDU. Comme UPEDU est un sous-ensemble de RUP, certains allégements se retrouvent. Le processus UPEDU présente l'aspect dynamique ainsi que l'aspect statique. Concernant l'aspect dynamique, nous retrouvons le caractère itératif du processus. Concernant l'aspect statique, six disciplines composent le modèle du processus. Chaque discipline est constituée par le regroupement logique des activités susceptibles d'être réalisées. Le résultat des activités est un artefact ou un ensemble spécifique d'artefacts. Par rapport au RUP, UPEDU propose un nombre sensiblement plus petit d'artefacts à produire au long du déroulement d'un projet. Enfin, plusieurs commentaires sur l'utilisation du processus UPEDU se retrouvent dans la littérature (Halling, Zuser, Kohle et Biffl, 2002).

Le deuxième processus utilisé dans le cadre de l'Atelier 2002 est le processus XP-A, qui est un processus basé sur XP. Les caractéristiques générales du processus XP ont été présentées dans le chapitre précédent. Les particularités, les avantages ainsi que les désavantages de l'application d'un processus XP dans un environnement éducationnel sont largement présentés dans la littérature (Williams, Kessler, Cunningham et Jeffries, 2000; Williams et Upchurch, 2001; Schneider et Johnston, 2003 ; Hedin, Bendix, et Magnusson, 2004). Le processus XP-A respecte les caractéristiques générales du processus XP, mais allégés et adaptées au cadre du cours Atelier Génie Logiciel.

UPEDU et XP sont deux processus qui utilisent des approches différentes pour communiquer l'information tout au long du déroulement d'un projet logiciel. Dans le but de pouvoir comparer les deux processus et donc de répondre à nos questions de

recherche nous avons fait un alignement tant pour l'aspect dynamique que pour l'aspect statique de processus.

Alignement de l'aspect dynamique

Bien que UPEDU et XP soient tous les deux des processus itératifs, la taille moyenne d'une itération varie selon le processus utilisé. Nous avons structuré la durée des itérations pour chacun des processus utilisés et nous avons adapté le cycle de vie logiciel avec des jalons qui sont alignés pour les deux processus. Les détails de cet alignement sont présentés par Germain et al. (2003). La figure 2.1 présente la structure des itérations pour les deux processus : UPEDU et XP

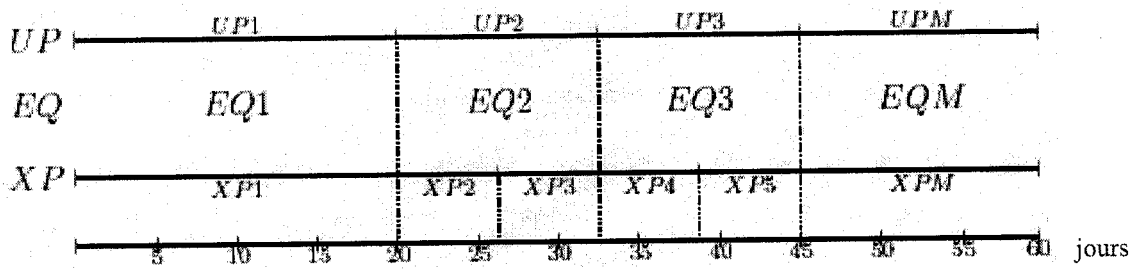

Figure 2.1. Alignement des itérations pour les processus UPEDU et XP

Le cycle de développement initial (45 jours) est composé de trois itérations UPEDU qui correspondent aux cinq itérations XP et se termine avec une première version du produit logiciel. Le cycle de développement est suivi par un court cycle de maintenance (15 jours) composé d'une seule itération pour les deux processus et à la fin duquel résulte la version finale du produit.

Alignement de l'aspect statique

Les éditions précédentes de l'Atelier Génie Logiciel ont montré que certains des activités définies par le processus UPEDU n'ont pas été correctement comprises par les étudiants (Germain, Dulipovici et Robillard, 2002a). Pour corriger cet aspect et surtout dans le but d'identifier les aspects cognitifs spécifiques au deux processus et donc de pouvoir les comparer correctement, nous avons pris en considération les activités cognitives que les développeurs exécutent d'habitude durant un processus logiciel. De plus, ces activités sont indépendantes du processus utilisé. En conséquence, nous avons développé une classification d'activités cognitives, tel que montrée dans le Tableau 2.1.

Tableau 2.1 Activités cognitives

Activités cognitives	Catégorie
Read Lire	Production
Écrire	
Dessiner	
Réfléchir	
Discuter	
Naviguer / Chercher	
Coder	
Coder & Tester (entrelacé)	
Tester	
Intégrer & Tester	
Inspecter / Réviser	Contrôle
Formation	Support

Une description détaillée de toutes les activités cognitives que nous avons utilisées est présentée par Germain et Robillard (2004a).

Toujours dans le but de pouvoir analyser les aspects cognitifs spécifiques aux deux processus, nous avons établi la liste des artefacts à produire dans le cas de chaque processus. Nous avons défini six groupes d'artefacts – quatre groupes pour l'aspect technique du développement de logiciel et deux groupes pour les aspects gestion du développement de logiciel.

Tableau 2.2 Correspondance des artefacts

		UPEDU	XP-A
Technique	G1	Spécifications des requis logiciel Glossaire de termes Modèle de cas d'utilisation	Spécifications des requis logiciel Histoire utilisateur
	G2	Réalisation des cas d'utilisation Modèle de conception	Métaphores Carte CRC Croquis UML
	G3	Composantes Modèle d'implémentation	Code unitaire Sous système
	G4	Plan de test Cas de test Résultats de test	Test unitaire Données de test Résultats de test
Gestion	G5	Plan d'itération Plan de développement Plan de mesures	Estimation des histoires Plan d'itération
	G6	Plan de gestion de configuration	Gestion de code

Étant donné que les artefacts de gestion de projet et de gestion du code du processus XP-A ne sont pas assez consistants pour former un groupe distinct chacun, nous avons regroupé ces artefacts dans un seul groupe – le groupe des artefacts de gestion. Le tableau de correspondance des artefacts propres à chaque processus est présenté dans le Tableau 2.2.

2.2.2. Le produit

Le produit à développer dans le cadre de l'Atelier Génie Logiciel 2002 est un outil de planification de réunions, téléphonique ou via Internet, utilisant le Web comme support de communication. Le logiciel REPLAN (Réunion Planification) doit permettre à des groupes de personnes de planifier des réunions en utilisant une base de données commune contenant les disponibilités de chacune des personnes. Le grand nombre d'intervenants dans une réunion rend nécessaire l'utilisation d'un modèle graphique de représentation de l'information requise aux fins de planification. La conjonction des horaires de tous les intervenants risque fort de ne laisser qu'un nombre restreint de périodes très courtes en guise de consensus. Un outil spécial de calendrier permettrait à un coordonnateur de visualiser les informations appropriées d'un coup d'oeil, rendant la décision de planification plus facile. Les décisions sont alors transmises électroniquement à tous les participants.

Le produit a été réalisé avec la famille Java/Servlet/JSP comme technique de base pour le développement. Deux serveurs informatiques ont été fournis : le premier – pour accueillir tant un système de fichiers qu'un Serveur d'Application Tomcat et le

deuxième – pour accueillir un serveur de base de données pour les applications des étudiants. Dans le laboratoire dédié à l'atelier nous avons installé un environnement de développement Java, un client de base de données et tout le logiciel nécessaire sur chaque station de travail.

2.2.3. Les sujets

Les sujets de notre étude observatoire sont les étudiants inscrits au cours Atelier Génie Logiciel, édition 2002. Tous les étudiants inscrits à ce cours avaient les prérequis demandées. Le premier jour de classe, nous avons demandé aux étudiants de former les équipes. La classe s'est divisée en six équipes (cinq d'entre elles étant composé de quatre étudiants et la sixième de trois étudiants). Chaque moitié de la classe a été assignée à un des deux processus, ainsi trois équipes (X1, X2, X3) ont été assignées à un processus basé sur XP et trois équipes (U1, U2, U3) ont été assignées au processus UPEDU. Une mention doit être faite concernant l'équipe U3 composée de trois étudiants. Les étudiants des équipes avec le processus UPEDU avaient suivi au préalable un cours spécifiquement sur ce processus.

2.2.4. La collecte des données

Il est largement reconnu que les aspects cognitifs sont difficiles à mesurer. En outre, le contexte académique de notre recherche ajoute des particularités à la cueillette des données. Nos analyses se basent sur l'effort mis par les étudiants – en tant que développeurs, sur chaque activité ou artefact produit.

Il est souvent dit qu'une image vaut mille mots. Dans le contexte des mesures de logiciels, cet aphorisme pourrait être interprété comme « une mesure valable vaut mille suppositions ». Les difficultés rencontrées dans les précédentes sessions de l'Atelier sont présentées par Germain et al. (2003).

Pour l'Atelier 2002, un outil industriel de gestion de temps a été utilisé dans le but de renforcer le processus de cueillette de données. Le système de jetons nous permet de comptabiliser l'emploi du temps de chacun des membres des équipes de développement afin de déterminer l'effort requis par les activités et les artéfacts de chacun des deux processus. Outre cela, tous les artefacts réalisés par chaque équipe ont été enregistrés dans un repositoire à part et ont été soigneusement analysés.

2.3. La structure du mémoire

Nos recherches sont basées sur les résultats trouvés dans la littérature, tel que présentés dans le chapitre 2 de ce mémoire. De plus, le cadre offert par le cours Atelier Génie Logiciel nous a permis de vérifier certains aspects spécifiques au processus de développement de logiciels. En réponse à nos deux questions de recherche, les résultats de nos études empiriques observatoires menés dans le contexte de l'Atelier 2002 sont présentés dans les chapitres suivants qui constituent le corps du travail de recherche.

Le chapitre 3 présente une réponse à la première question de recherche, soit de déterminer l'influence possible du processus de développement sur la qualité de réalisation des requis. Ce sujet est exposé par l'article intitulé « L'influence du processus sur la qualité du produit », auteurs Mihaela Dulipovici et Pierre-N. Robillard. L'article a été accepté et il a été publié dans la revue « Ingénierie des Systèmes d'Information », le numéro spécial « Information Systems Quality », janvier 2005, volume 9 / 5-6, p. 103-116. Les résultats présentés dans l'article indiquent la possibilité d'un lien entre la qualité de la réalisation des requis et le processus utilisé pour développer le produit logiciel. Les indices fournis peuvent être utiles dans l'évaluation des processus de développement et leur influence sur la qualité de logiciel.

Le chapitre 4 présente une réponse à la deuxième question de recherche, soit d'examiner les aspects cognitifs lors de la création des artéfacts tout au long d'un processus logiciel. Ce sujet est exposé par l'article intitulé « Cognitive Aspects in a Project-Based Course in Software Engineering », auteur Mihaela Dulipovici et Pierre-N. Robillard. Une étude préliminaire a été présentée à la Conférence ITHET 2004 (The 5th

International Conference on Information Technology Based Higher Education). Le sujet a été sélectionné en vue de publication par la revue IEEE Transactions on Education. L'article dans sa version finale a été soumis à la revue en avril 2005. L'étude observatoire est basée sur une approche originale permettant à la fois d'identifier et d'analyser les aspects cognitifs propres à la création des artefacts. Outre cela, un modèle à trois pôles est proposé. Les résultats présentés dans l'article permettent une meilleure compréhension du comportement cognitif des développeurs afin de proposer des actions en vue d'améliorer le processus logiciel.

Des discussions sur le sujet, les limites de nos recherches, qui sont inhérentes à toute étude empirique, les leçons apprises ainsi que des pistes pour continuer les recherches sont présentées dans le chapitre Discussion générale et conclusions.

Par ailleurs, les résumés des quatre articles de conférence auxquels l'auteur a contribué sont inclus dans l'annexe. Les deux premiers articles (Germain, Dulipovici, Robillard 2002a; Germain, Robillard, Dulipovici, 2002b) portent sur l'édition 2001 de l'Atelier et montre la faisabilité de l'approche entreprise dans la présente recherche. Ces articles illustrent également que certaines activités cognitives de processus sont difficiles à dissocier et soulèvent le fait de l'ambiguïté et de la confusion de compréhension des activités. Le troisième article (Germain, Dulipovici, Cherry, Robillard, 2003) présente les difficultés qui souvent rendent difficile la cueillette de données – fait qui influence la véridicité des résultats et suggère certains solutions possibles. Le quatrième article (Dulipovici, Robillard, 2004a) présente une étude préliminaire sur les aspects cognitifs lors de la création des artefacts dans le cadre d'un cours-projet en génie logiciel.

CHAPITRE 3

L'INFLUENCE DU PROCESSUS SUR LA QUALITÉ DU PRODUIT

Résumé

Cet article présente une étude réalisée en milieu universitaire pour déterminer l'influence possible du processus de développement sur la qualité de réalisation des besoins. Les mêmes besoins ont servi à six équipes de développeurs logiciels. Trois équipes ont utilisé le processus structuré UPEDU dérivé du RUP et trois équipes ont utilisé un processus de type agile – XP. La qualité de réalisation de chacun des besoins a été évaluée par un comité d'experts. Sur les 109 besoins réalisés il semble que la qualité de 46 d'entre eux puisse dépendre du processus utilisé. De plus, il semble que la qualité de 27 de ces besoins soit fonction des méthodes de conception utilisées dans le cadre du processus. L'article fournit des indices qui peuvent être utiles dans l'évaluation des processus de développement et leur influence sur la qualité de logiciel.

Abstract

This paper presents the results of an observational study carried out in an academic environment in order to determine the possible influence of the development process on the quality of software requirements realization. All six teams developed their own version of the system based on the common specification. Three teams used the

structured process UPEDU derived of the RUP and three teams used an agile type process – XP. An experts' committee evaluated the realization quality of each requirement. On 109 realized requirements it seems that the quality of 46 of them can depend on the used process. Furthermore, it seems that the realization quality of 27 of these requirements is function of the design methods used within the framework of the process. The article supplies indications that can be useful in the evaluation of the development processes and their influence on the quality of the final software product.

MOTS-CLÉS : qualité du produit logiciel, besoins logiciel, processus de développement, processus unifié, méthodes agiles, recherche empirique

KEYWORDS: software product quality, software requirements, development process, unified process, agile methods, empirical research

3.1. Introduction

Dans le monde fortement compétitif d'aujourd'hui et à la vitesse du Web, produire un logiciel de qualité est sans cesse un défi et un sujet d'intérêts pour les développeurs de logiciel. La qualité est un concept élusif et, d'acceptation générale, elle est aussi une notion composée. Kitchenham et Pfleeger (1996) affirment « la qualité est difficile à définir, impossible à mesurer, mais facile à reconnaître ». Garvin (1984) présente plusieurs perspectives de la qualité du logiciel et montre comment elles influencent la définition de la qualité. De la perspective de l'utilisateur, la qualité peut être définie comme le degré auquel un client ou un utilisateur perçoit la façon dont le logiciel accède

aux attentes. Cette approche implique que la qualité du logiciel n'est pas absolue, mais qu'elle peut varier avec le temps et les attentes des utilisateurs. Usrey et Dooley (1996) synthétisent cette idée dans une définition très suggestive : « la qualité se trouve dans l'oeil de l'utilisateur ». Cette définition de la qualité réfère au degré auquel un système logiciel atteint les besoins de l'utilisateur. Dans cette perspective, tous les aspects reliés aux fonctionnalités du système logiciel sont de la plus grande importance pour l'utilisateur.

Le développement logiciel est considéré comme composé de trois éléments de base: le produit, le processus et les développeurs. Depuis quelques années, le processus est perçu comme un élément important pour améliorer la qualité du produit logiciel. Un processus logiciel est la définition des relations qui existent entre les activités, les rôles et les artefacts nécessaires à la réalisation d'un projet logiciel (Robillard, Kruchten et d'Astous, 2002). Cependant, il n'existe pas encore de processus universellement reconnu. La méthodologie de développement logiciel basée sur le triplet « activité – rôle – artefact » est une approche systématique pour la capture des besoins, en utilisant des scénarios détaillés, des diagrammes divers, et en rédigeant un modèle de cas d'utilisation. Il y a aussi un glossaire de termes, des requis supplémentaires, des contraintes de conception, des besoins d'interface, et ainsi de suite. Tous ces artefacts capturent les besoins du système à développer et par conséquent – les besoins de l'utilisateur. Le document de spécification des besoins est le contrat détaillé sur lequel se base la conception, l'implémentation, les tests ainsi que la plupart des activités de l'équipe de développeurs. La méthodologie basée sur le triplet « activité – rôle –

artefact » souligne l'importance des artefacts écrits pour capturer les besoins, pour présenter les décisions de conception d'architecture ou pour la planification d'essais.

En réaction à ce type de méthodologie et particulièrement à la grande quantité de documentation écrite requise, il y a, depuis quelques années, un intérêt croissant pour les méthodologies dites de type agiles (Manifesto for Agile Software Development – Agile Alliance, 2004). Ces méthodologies plus souples visent à faciliter l'intégration des changements qui peuvent survenir pendant toutes les étapes du processus de développement. XP (Extreme Programming) (Beck, 1999), ASD (Adaptive Software Development) (Highsmith, 2000), Scrum (Rising and Janoff, 2000), Crystal (Cockburn, 2002; Highsmith, 2002) sont quelques exemples de méthodologies de types agiles. Parmi toutes ces méthodologies, XP (Extreme Programming) (Beck, 1999, 2000) est celle qui a reçu la plus grande attention. Dans le cas d'un processus de type XP le client fait partie de l'équipe de développement et il est responsable de définir les besoins initiaux du système à développer. Les fonctionnalités exigées par le système sont capturées dans des scénarios très courts, qui sont ensuite décomposés en de petites tâches. L'analyse de ces besoins est réalisée au cours de conversations entre le client-utilisateur et l'équipe de développement. La méthodologie de type agile fait de la communication verbale une de ses valeurs principales et propose une approche minimaliste à la documentation écrite.

Bien qu'il soit reconnu qu'un choix approprié d'un processus de développement peut améliorer la qualité du produit, il y a peu d'évidence empirique sur les liaisons directes entre le processus et la qualité de réalisation des besoins du produit logiciel. Le but de

nos recherches est de déterminer l'influence possible du processus de développement sur la qualité de réalisation des besoins. Cet article présente les résultats d'une étude d'observation effectuée dans le contexte de l'Atelier de Génie Logiciel – un cours à projet pour des étudiants en fin d'étude à l'École Polytechnique de Montréal. Les sections de l'article présentent les particularités du contexte dans lequel le travail de recherche a été réalisé, l'analyse des données et une discussion des résultats obtenus. Les conclusions peuvent permettre une meilleure compréhension du rôle joué par un processus de développement spécifique et son importance pour la qualité du produit logiciel final.

3.2. Le contexte du projet

Cet effort de recherche a été inspiré par plusieurs études précédentes entreprises dans le contexte de l'Atelier de Génie Logiciel à l'École Polytechnique de Montréal. L'Atelier est en même temps un cours à un semestre pour des étudiants de niveau senior et une plate-forme de recherche pour les aspects spécifiques de génie logiciel. Les étudiants inscrits dans ce cours doivent développer durant la session un système logiciel complet et fonctionnel, basé sur une spécification de besoins fournie, et en respectant un processus logiciel bien précis. Les projets de l'Atelier sont basés sur l'utilisation du UPEDU (Unified Process for Education) (Robillard, Kruchten et d'Astous, 2002) qui est une version allégée du RUP (Rational Unified Process) (Kruchten, 2000) adaptée à un environnement universitaire, dans le but d'enseigner aux étudiants les règles et les pratiques de développement d'un produit logiciel. Toutes les équipes d'étudiants sont en

compétition sur le même projet, et développent leur propre version du système mais basée sur une spécification des besoins unique.

La structure de ce type de cours-projet est présentée en détail par Robillard (1996). Les résultats de plusieurs études sur la répartition de l'effort de développement logiciel, des activités exécutées et la qualité des artefacts ont été présentés par (Robillard, 1998), (Germain, Dulipovici et Robillard, 2002a, 2002b), (Germain et Robillard 2004a), (Dulipovici et Robillard, 2004a).

Le contexte de l'Atelier 2002 est unique par l'utilisation de deux processus différents: UPEDU (Unified Process for Education) et un processus basé sur XP (Extreme Programming). UPEDU et XP fournissent deux approches différentes aux projets de développement de logiciel. UPEDU est un processus logiciel complet qui définit toutes les activités et les artefacts nécessaires pour réaliser le projet dans le cadre de l'Atelier. XP est également un processus qui présente des activités et des artefacts différents par rapport aux pratiques traditionnelles. Bien que UPEDU et XP soient tous les deux des processus itératifs, la taille moyenne d'une itération varie selon le processus utilisé. Nous avons structuré la durée des itérations pour chacun des processus utilisés et nous avons adapté le cycle de vie logiciel avec des jalons alignés pour les deux processus.

Le cours-projet comprend six équipes dont trois équipes (X1, X2, X3) qui utilisent le processus basé sur XP et trois équipes (U1, U2, U3) qui utilisent le processus UPEDU. Tous les étudiants inscrits au cours-projet avaient les prérequis nécessaires et les compétences semblaient assez uniformes parmi les équipes. Les étudiants des

équipes avec le processus UPEDU avaient suivi au préalable un cours spécifiquement sur ce processus.

Le produit à développer dans le cadre de l'Atelier Génie Logiciel est un outil de planification de réunions, téléphonique ou via Internet, utilisant le Web comme support de communication. Le logiciel REPLAN (Réunion Planification) doit permettre à des groupes de personnes de planifier des réunions en utilisant une base de données commune contenant les disponibilités de chacune des personnes. Le grand nombre d'intervenants dans une réunion rend nécessaire l'utilisation d'un modèle graphique de représentation de l'information requise aux fins de planification. La conjonction des horaires de tous les intervenants risque fort de ne laisser qu'un nombre restreint de périodes très courtes en guise de consensus. Un outil spécial de calendrier permettrait à un coordonnateur de visualiser les informations appropriées d'un coup d'oeil, rendant la décision de planification plus facile. Les décisions sont alors transmises électroniquement à tous les participants. Une copie d'écran d'un produit final est présentée à la Figure 3.1.

Le produit a été réalisé avec la famille Java/Servlet/JSP comme technique de base pour le développement. Les étudiants devaient respecter les besoins et ne pas ajouter de fonctionnalités non explicitement demandées.

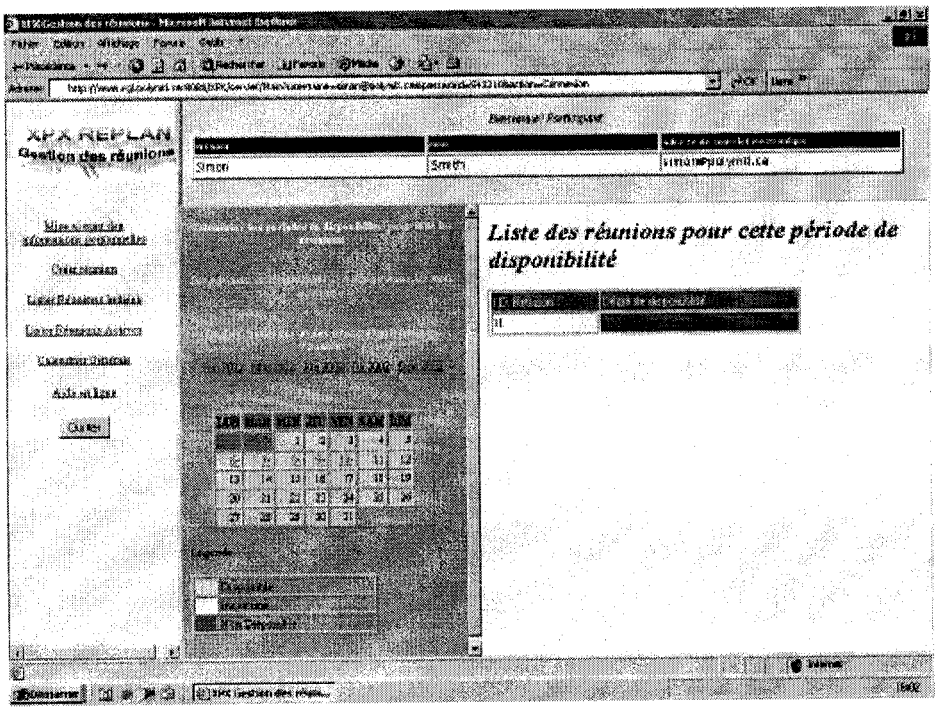

Figure 3.1 Capture d'écran d'un produit REPLAN

3.3. Le contexte de recherche

Le but de cette étude est de déterminer l'influence du processus de développement de logiciel sur la qualité de réalisation des besoins du logiciel. Nous avons analysé tous les produits finaux en comparant les résultats des réalisations des besoins par rapport aux besoins spécifiés. Tous les produits logiciels ont été exécutés à partir de la même base de tests d'acceptation et la réalisation de chaque fonctionnalité a été jugée par un comité d'experts. L'hypothèse de base de la méthode de mesure proposée est que les experts logiciels peuvent faire des jugements logiques sur les niveaux d'accomplissement liés aux divers éléments ou attributs de qualité. Nous avons utilisé une échelle à trois

niveaux pour évaluer quantitativement les jugements des experts, comme indiqués dans le Tableau 3.1.

Nous avons observé qu'il n'y a aucune asymétrie significative concernant la réalisation des besoins parmi les équipes, particulièrement parmi les trois équipes utilisant le même processus pour le développement de logiciel. Basé sur cette observation et le fait que toutes les équipes ont développé leur produit basé sur la même spécification de besoins, nous avons fait la supposition que les points obtenus par chaque équipe ont la même pondération et peuvent ainsi être concaténés. Pour chaque fonction indiquée dans la spécification des requis, chaque groupe de trois équipes, les groupes UPEDU (U) ou les groupes XP (X) utilisant le même processus peut obtenir un vecteur de points maximal de 2-2-2. Ce qui signifie, par exemple, que les équipes U1, U2 et U3 ont tous obtenu 2 points pour l'évaluation d'un besoin z.

Tableau 3.1. Échelle de quantification des jugements des experts

Échelle de préférence	Définition
2	Besoin réalisé complet et conforme aux spécifications
1	Besoin partiellement réalisé par rapport aux spécifications
0	Besoin non réalisé ou le résultat n'est pas celui attendu

Généralement, une fonctionnalité est traitée comme une propriété d'un système, souvent comme une abstraction du comportement du système. L'utilisateur a un but à l'esprit qu'il voudrait que le système accomplisse. Dans le développement de logiciel, un résultat possible est une attente incohérente, c'est-à-dire, qu'il existe une différence d'une part – entre la fonctionnalité que les développeurs croient devoir réaliser – et d'autre part la fonctionnalité que les utilisateurs croient devoir obtenir. La différence principale entre les deux processus utilisés, UPEDU et XP est la voie recommandée pour l'explication, la clarification et l'exposition des détails concernant chacun des besoins entre les membres d'équipe de développement ainsi qu'à l'extérieur de l'équipe.

UPEDU est un processus basé sur les cas d'utilisation. Un cas d'utilisation définit un ensemble d'interactions entre des acteurs externes et le système à l'étude. Il décrit l'ordre des interactions entre les acteurs et le système qui est nécessaire pour livrer un service susceptible de satisfaire l'utilisateur. Une représentation courante des interactions utilisateur – système est le scénario, qui est écrit en utilisant le vocabulaire du domaine d'application, d'une façon bien structurée et compréhensible par l'utilisateur. Un scénario, en tant qu'instance d'un cas d'utilisation, capture la série d'interactions utilisateur – système nécessaire à la réalisation de requis. Les scénarios peuvent être représentés graphiquement en utilisant des diagrammes UML. Un modèle de cas d'utilisation consiste en la totalité des acteurs du système ainsi qu'en la totalité des cas d'utilisation différents par lesquels les acteurs agissent réciproquement avec le système. Le modèle décrit ainsi la totalité du comportement fonctionnel du système. (Robillard, Kruchten et d'Astous, 2002).

XP définit une « histoire d'utilisateur » comme la quantité la plus petite d'informations exigée pour permettre à l'utilisateur de définir un chemin du système. Une « histoire d'utilisateur » est une courte description d'une certaine partie d'un besoin. Elle contient juste assez de texte pour expliquer ce qui doit être fait pour que l'utilisateur et les développeurs puissent comprendre de la même manière l'élément du besoin. Ce n'est pas un grand document, mais plutôt un paragraphe simple et facile à comprendre. Cependant, il est toujours possible que des développeurs et l'utilisateur aient une compréhension différente d'une « histoire d'utilisateur ». Pour clarifier ce malentendu, ils discutent, la communication verbale étant une des valeurs principales de l'approche *Extreme Programming*. Comme le Manifeste Agile précise « la méthode la plus efficiente et efficace de transmettre des informations parmi les membres d'une équipe de développement ainsi qu'à l'extérieur de l'équipe est la conversation face à face ». (Beck, 2000).

3.4. Analyse et résultats

Il est largement accepté qu'un ensemble clair de besoins facilite la qualité d'un système. La spécification des besoins inclut la description précise de fonctionnalités nécessaires et des interactions exigées entre l'utilisateur et le système. Les 109 besoins exigés pour le produit REPLAN ont été clairement présentés dans un document qui constituait la base de référence pour toutes les équipes de développement. Cette liste de besoins était complète et suffisante et aucun autre besoin n'a été ajouté ou modifié en cours de projet. Le Tableau 3.2, présente l'évaluation globale pour chacune des équipes

en termes de points obtenus pour les fonctionnalités réalisées ainsi que la valeur relative par rapport au maximum possible. Ce résultat est basé sur les jugements d'experts selon une échelle de 0 à 2 pour chacun des 109 requis.

Tableau 3.2. Résultats totaux des équipes

Équipes	UPEDU			XP		
	U1	U2	U3	X1	X2	X3
Résultat total (valeur absolue)	178	130	156	164	126	119
Résultat total (valeur relative)	82%	60%	72%	75%	58%	55%

Chaque type de processus a une équipe « championne » avec un meilleur résultat pour le degré de réalisation des besoins, mais aucune équipe n'a atteint le résultat maximal. Il n'y a aucun produit REPLAN avec une réalisation parfaite des besoins.

En analysant la structure des points des équipes, nous remarquons qu'un nombre plus grand de besoins réalisés avec succès ne signifie pas nécessairement un résultat final meilleur. Après la comparaison de la structure des points de chaque équipe avec les équipes du même type de processus ainsi qu'avec les équipes de l'autre type de processus, nous concluons que ces résultats généraux n'ont pas montré un lien direct entre le processus utilisé et la qualité de réalisation des besoins. La structure des points à elle seule ne peut montrer si certains types de besoins sont mieux réalisés que d'autres.

Basé sur la quantification des jugements des experts, chaque besoin est associé à un vecteur de points à trois dimensions. Deux catégories de besoins ont été identifiées :

– les besoins avec le même degré de réalisation par les six équipes → les besoins avec les vecteurs de points associés [2-2-2 – 2-2-2], [1-1-1 – 1-1-1], [0-0-0 – 0-0-0]

– les besoins avec un degré différent de réalisation pour un type de processus → les besoins avec les vecteurs [2-1-2 – 2-2-1], [0-0-1– 0-1-0]

Comme l'objectif de cette étude est d'identifier la différence que peut engendrer un processus sur la qualité du produit, nous avons dans un premier temps enlevé les besoins qui ont des vecteurs identiques pour les deux types de processus. En d'autres mots, ces besoins ont été parfaitement, partiellement ou aucunement réalisés par chacune des équipes indépendamment du processus utilisé.

Parmi les fonctions avec un degré différent de réalisation par les six équipes, nous identifions comme équivalent les besoins dont le vecteur de points a les mêmes données sans égard à leur ordre dans le vecteur. Cette l'hypothèse est basée sur le fait que la position de l'équipe dans le vecteur de point est arbitraire. Par exemple, les vecteurs de points [2-1-1]; [1-2-1]; [1-1-2] sont d'une valeur égale pour notre d'analyse. Tous les besoins ayant ce même profil d'évaluation dans les deux types de processus (UPEDU et XP) sont également retirés de l'analyse.

Suite à ces deux phases de sélection, il reste 46 besoins qui constituent la conséquence possible de l'utilisation de deux processus différents. Pour certains besoins, la différence entre les deux vecteurs de points n'est pas si évidente. Au contraire, pour d'autres besoins, il y a des différences significatives entre les deux

vecteurs de points en faveur d'un type de processus ou d'un autre. Le Tableau 3.3 illustre quelques exemples.

Tableau 3.3. Exemples de différences entre les vecteurs de points

	UPEDU	XP
Différence non significative	2-2-1	2-2-2
	2-1-1	2-1-0
Différence significative	1-0-0	2-2-2
	2-2-2	1-0-0

Une analyse détaillée est réalisée sur le groupe de 27 besoins ayant une différence significative dans les vecteurs de points. La Figure 3.2 montre les vecteurs de points pour ces besoins dans le cas des produits logiciels développés en utilisant les processus UPEDU et XP. Avec peu d'exceptions, le vecteur de points montre une spécificité évidente dans la réalisation des besoins.

Nous observons clairement des besoins pour la réalisation desquels le processus UPEDU a obtenu plus de points et de l'autre côté, des besoins pour la réalisation desquels le processus XP a obtenu plus de points. Cette constatation indique la possibilité d'un lien entre la qualité de réalisation des besoins et le processus utilisé. Une analyse plus approfondie sur ces deux catégories de besoins tente de trouver quel est ce lien.

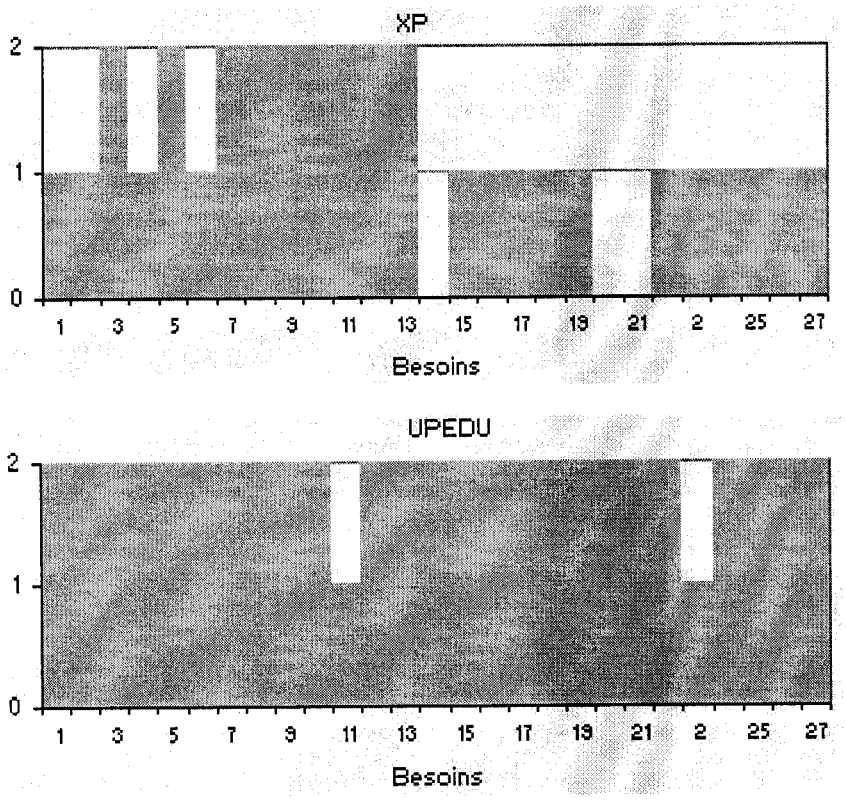

Figure 3.2 Vecteurs de points avec différence significative

3.5. L'agrégation des fonctionnalités

Les développeurs traitent les besoins comme les propriétés du produit logiciel. Du point de vue de l'utilisateur, les besoins semblent être des mots pour explorer les caractéristiques de leur produit logiciel. Chandrasekaran et Kaindl (1996) proposent une définition générale pour les besoins, qui montre la dualité du terme : « étant donné une fonction F, un objet O et un mode de déploiement M, nous pouvons dire que F est une fonction de O, s'il y a un mode de déploiement M pour que O sous le M occasionne les effets indiqués dans F dans les conditions associées ». L'utilisateur s'imagine un certain effet dans de certaines conditions et si un objet peut créer l'effet, dans ce cas il peut

attribuer l'effet comme une fonctionnalité de l'objet, soit un besoin. D'une part, les fonctionnalités peuvent résulter des activités dynamiques de l'objet, c'est-à-dire, des changements d'état – des besoins dynamiques. D'autre part, il y a des objets qui réalisent leur fonctionnalité simplement en raison de leurs propriétés structurelles – des besoins statiques.

Le logiciel REPLAN visé par le projet sert à combler les besoins d'un produit logiciel pour la tenue de réunions physiques ou virtuelles, tel que de :

- visualiser l'ensemble des informations sur les disponibilités des intervenants dans une réunion
- mesurer le degré de pertinence de la tenue d'une réunion à différentes périodes, calculé à partir d'une fonction d'agrégation
- faire la mise à jour des informations et les enregistrer dans une base de données unique
- visualiser les conséquences des choix possibles pour la tenue d'une réunion

Les besoins du produit REPLAN peuvent être regroupées autour des trois entités principales, soit l'entité « Réunion », l'entité « Client-Initiateur » ainsi que l'entité « Client-Participant ».

L'entité « Réunion » est une structure d'information regroupant les attributs d'une réunion physique ou virtuelle entre plusieurs individus. En terme orienté-objet, une « Réunion » est un objet avec plusieurs attributs, parmi lesquels :

- la plage de dates et d'heures possibles pour la réunion
- le lieu dépendant du type de la réunion

– la liste des participants potentiels avec des informations spécifiques pour chaque combinaison réunion - participant

L'entité « Client-Initiateur » regroupe les personnes qui ont le droit de planifier des réunions et de lancer des invitations à ces réunions. Les opérations de l'objet « Client-Initiateur » sont :

– créer une instance de réunion

– choisir les informations spécifiques à la réunion

– inscrire la liste des invités et définir le degré de criticité de leur présence à la réunion

– prendre connaissance des disponibilités des participants

– finaliser la convocation des réunions en tenant compte de toutes les contraintes

L'entité « Client-Participant » regroupe les personnes qui peuvent être invitées à des réunions. Les opérations de l'objet « Client-Initiateur » sont :

– s'informer sur le contenu et la tenue des réunions

– inscrire les disponibilités

– accepter ou non une convocation à une réunion

Une première remarque concernant les trois entités du produit REPLAN est évidente: pour l'entité « Réunion » les informations les plus importantes portent sur l'état de l'objet, tandis que pour les entités « Client-Initiateur » et « Client-Participant » les informations les plus importantes portent sur le comportement de l'objet respectif. Nous regroupons l'ensemble des fonctionnalités du système REPLAN autour de ces trois

entités. Les points totaux marqués pour chaque entité par les équipes des deux types de processus sont présentés dans la Figure 3.3.

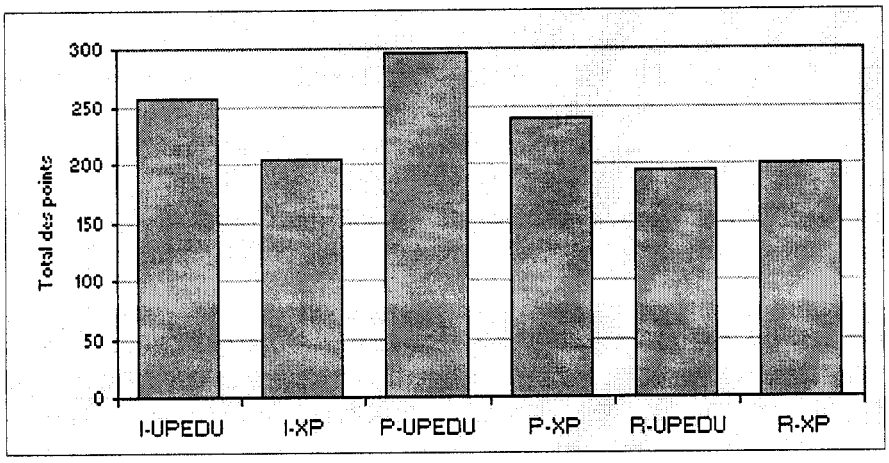

Figure 3.3 Comparaison des points pour les trois entités de REPLAN (I-Initiateur, P-Participant, R-Réunion)

Les profils de la réalisation de besoins pour les trois entités montrent que les équipes UPEDU ont une meilleure réalisation des entités « Client-Initiateur » et des entités « Client-Participant ». Toutefois, les équipes XP font une réalisation aussi bonne de l'entité « Réunion ».

Pour les deux entités comprenant des besoins dynamiques, les équipes utilisant UPEDU ont obtenu un résultat meilleur pour la qualité de la réalisation des fonctions que les équipes utilisant XP. Les requis dynamiques visent le comportement du système. Des scénarios détaillés, des diagrammes de cas d'utilisation, de séquence, de collaboration et d'état semblent plus appropriés pour représenter les détails de chaque fonction. La représentation schématique semble faciliter le travail. D'ailleurs, pour l'entité « Réunion » qui comprend seulement des fonctions statiques, les équipes

utilisant XP ont obtenu un résultat légèrement meilleur pour la qualité de la réalisation de ces besoins que les équipes utilisant UPEDU. Pour ce type de besoins, les histoires succinctes d'utilisateur sont probablement suffisantes car tous les détails nécessaires sont ajoutés à la suite de la communication directe entre l'utilisateur et les développeurs.

3.6. Conclusion

La qualité est un concept à facettes multiples et qui peut être interprété différemment d'une perspective à l'autre. Pour réaliser un haut niveau de qualité, il est essentiel que les besoins du système soient développés correctement et complètement. De plus en plus, le processus de développement joue un rôle important dans la qualité des produits du logiciel. Le choix du processus s'avère très difficile à cause d'un grand nombre de facteurs à considérer.

Cette étude examine les liens entre le processus de développement et la qualité de réalisation des besoins d'un produit logiciel. Une étude observationnelle est effectuée dans le contexte de l'Atelier de génie logiciel, un cours à un semestre, orienté projet pour des étudiants de niveau fin d'études à l'École Polytechnique de Montréal. Nous analysons la qualité et le degré de réalisation des besoins pour six produits logiciels basés sur les mêmes spécifications, mais qui ont été développés en respectant deux processus différents. Les processus utilisés dans cette étude sont: UPEDU (Processus Unifié pour Éducation) – une version allégée du RUP (Rational Unified Process) et adaptée à un environnement universitaire et XP – le membre bien connu des méthodologies agiles.

Les résultats de cette étude indiquent la possibilité d'un lien entre la qualité de la réalisation des besoins et le processus utilisé pour développer le produit logiciel. 42% des besoins analysés indiquent l'existence de ce lien. 58% des besoins de ce groupe indiquent que la qualité est fonction des méthodes de conception utilisées dans le cadre du processus. Selon les données, la méthodologie de développement basée sur le triplet activité – rôle – artefact, qui souligne l'importance des artefacts écrits pour capturer les besoins du système semble plus appropriée pour développer des entités basées sur des besoins à caractère dynamique. D'autre part, la méthodologie de développement agile qui capture les besoins du système par communication verbale entre le client et l'équipe de développement semble plus appropriée pour développer des entités basées sur des besoins à caractère statique. Les artefacts de modélisation non-UML semblent suffisants pour cette catégorie de besoins. Tous les détails du développement d'interface utilisateur ou de la modélisation de données sont mieux clarifiés par la communication directe.

Les études en génie logiciel conduites dans un environnement universitaire et utilisant des étudiants comme sujets, apportent une contribution valable au domaine des mesures empiriques en génie logiciel. L'étude présentée dans cet article fournit des indices qui peuvent être utiles en évaluant des processus de développement et leur impact sur la qualité de logiciel. Ces résultats peuvent permettre une meilleure compréhension du rôle joué par un processus spécifique ainsi que prouver une fois de plus son importance dans la qualité du produit logiciel final. Cependant, nous devons être prudents dans l'application de ces résultats en milieu professionnel en raison de la nature universitaire du projet, du cycle de vie particulier et de la technologie choisie.

Remerciements

Nous sommes reconnaissants à Éric Germain qui a été fortement impliqué dans la définition des besoins du système REPLAN et qui a agi en tant qu'instructeur pour le cours. Nous voudrions aussi remercier Martin Robillard pour ses commentaires perspicaces lors de la définition des besoins. Ce projet n'aurait pas été possible sans la participation volontaire et enthousiasme des étudiants inscrits au cours Atelier de génie logiciel, session hiver 2002 à l'École Polytechnique de Montréal.

Ce travail a été en partie soutenu par le Conseil National de la Recherche Scientifique de Canada (CNRSC) sous la subvention A0141.

CHAPITRE 4

COGNITIVE ASPECTS IN A PROJECT-BASED COURSE IN SOFTWARE ENGINEERING

Abstract

This paper presents an original approach for identifying and analyzing cognitive aspects specific to artifacts creation in the context of a senior-level project-course. The effort spent by student teams was analyzed for each specific artifact with regard to two different processes carried on in parallel: UPEDU (Unified Process for Education) – a traditional activity-role-artifact based methodology and XP (Extreme Programming) – an agile methodology. For this purpose the concept of mental model was extended to software process and the terms mental artifact and physical artifact were proposed for consideration. The findings show the similarities and the differences in the patterns followed by students for mental artifacts – representing the tacit knowledge, as well as for physical artifacts – representing the explicit knowledge, in the context of both processes. Even though the two used processes are very different, the results show that the same effort is spent on mental artefacts as on physical artefacts, in the context of both processes. A three poles cognitive model is suggested for consideration. The space of three poles cognitive structure shows the evolution curve for artefacts creation in the case of the two used processes. The comparison results may allow a better

understanding of students' cognitive behavior, in order to determine the required actions for improving academic projects.

Index Terms – software engineering education, collaborative teams, interpersonal communication, cognitive aspects

4.1. Introduction

Software is the result of the knowledge acquisition activity which, in turns, involves the mental process of comprehension, understanding and learning. The software process development could be perceived as the communication activity through which the knowledge is brought together and embodied in software. This approach points to the importance of the careful selection, not only of the best practices for developing software, but also of those suitable for an academic environment. The result would be the increase of the knowledge acquisition efficiency.

However, there is no common acceptance of a unique process practices set. Two main software development philosophies have emerged (Kruchten, 2000; Cockburn, 2002): the traditional activity – role – artifact based methodology and the agile methodology.

The first philosophy promotes a discipline-based engineering process involving effective definition of the activities to be performed, artifacts to be produced and roles to be played. This methodology involves the production of artifacts to support early decision-making on requirements and design matters, effective communication and

knowledge transfer. The Rational Unifed Process (RUP) (Kruchten, 2000) is an example of a process that fits this approach. RUP involves requirements analysis, an "up-front design", test planning, scheduling and progress tracking, and capturing all of that in comprehensive written artifacts.

The second philosophy, called "Agile Software Development" (Cockburn, 2002), breaks with a number of traditional software engineering practices and prescribes some special practices instead. Some of the key concepts shared by the agile methods specifically downplay the importance of written documentation and emphasize the verbal communication. XP (Extreme Programming) (Cockburn, 2002) is the most prominent member of the family of agile development methodology.

Previous studies have demonstrated the advantages and the problems encountered by using either one methodology in an academic context. There is no common acceptance of the best way students have to communicate information during the software development. The two processes RUP and XP rely on polar opposite approaches to software development regarding to the medium of communicating information. The study reported here is unique by using in parallel, in the same project-based course two processes that provide two different approaches to software development. The analysis is based on artifacts – considered as a direct way of recording and communicating information and an efficient form of knowledge transfer.

Knowledge is an elusive notion, based on the paradox that it resides in a person mind and at the same time has to be captured, stored and represented. Two types of knowledge can be identified: the explicit knowledge (tangible issues – artifacts) and the

implicit knowledge (tacit knowledge). Mental models representing the tacit knowledge play a central role in learning and in improving problem-solving, as it is the case of software development. Through the mental model, the students can explain, infer and understand the problem in order to decide what action to take to control its execution. Mental model is an accepted concept in the human-computer interaction literature (Sasse, 1992). However, as Chen (1992) stresses "... mental models are important not only to the use of software but to the development of software as well."

Learning and knowledge sharing are inter-dependent (Nonaka, Takeuchi, 1995). At the individual level, people learn from personal experience and reflection. Shared mental models extend the notion of individual mental models to a team context (Levesque, Wilson et Wholey, 2001). For a same problem, each team member has his own mental model, which is more or less different than team colleague's mental model. Effective communication between team members allows the construction of a team mental model based on the individual models. Team-level learning proceeds by communicating and sharing new insights with teammates. One difficulty related to mental models is that they are not directly observable. Their measurement continues to be a challenging undertaking, given the fact that models often take multiple forms and are dynamic in nature.

To address these problems, it was developed a unique approach for observing and analysing the process of students' knowledge crystallization (Robillard, 1999), both explicit and implicit, during a project-oriented one-semester course in software engineering for senior level students. The effort made by student teams on mental and

physical artifacts was analyzed for each specific artifact intended for two different processes carried on in parallel. This paper presents the results of an observational study carried out in the context of the "Software Engineering Studio".

4.2. Research context

By using the results of previous research as a foundation (Germain, Robillard, Dulipovici,. 2002a), a studio-based teaching and learning approach was adopted for the observational study. According to Gagné (1985), for a better learning, two conditions have to be satisfied: the external conditions (the particular teaching environmental techniques for facilitating the learning) and the internal conditions (prerequisite knowledge or capabilities that a student must possess).

4.2.1. Teaching and Learning Environment

Regarding to the external conditions mentioned above, the development process, the project subject, the time record tool, and the logistic informatics environment were adapted for the purpose of the Studio. The teaching method is based mainly on teamwork, all teams being in competition on the same project. The course does not contain a final exam, but class presentations and a formal acceptance test session.

Process: The main feature of 2002 Studio is the use of two different software processes: UPEDU (Unified Process for Education) (Robillard, Kruchten, d'Astous, 2002), and a process derived from XP (Cockburn, 2002). UPEDU is a scaled-down version of the RUP adapted for an academic environment. UPEDU is built on a set of

focused activities with significant cognitive content, and on a set of essential artifacts that should be needed for small projects. XP is a lightweight, yet disciplined software development methodology.

The UPEDU framework is based on two dimensions. The first one represents the dynamic aspect of the process and is expressed in terms of phases and iterations. XP is also an iterative process, but in UPEDU the iterations are longer and fewer than in XP. A software lifecycle with aligned milestones for both processes was designed for this edition of the Studio.

The second dimension of UPEDU represents the static aspect of the process, and is expressed in terms of roles, artifacts and activities. This dimension is realized differently in XP. A cognitive activity classification independent of the process used was developed (Germain, Robillard, 2003) and an artifact mapping table, containing the particular artifact to be produced, was drawn up (Dulipovici, Robillard, 2004).

Project: The Winter 2002 edition of the Studio featured the development of a Web-based meeting management system aimed at organizers of meetings where the number and geographic dispersion of participants make scheduling difficult. The software system to be developed would allow meeting coordinators to send availability requests to a set of individuals so that each one can specify their personal availability periods. The set of availability periods would then be graphically represented using a special calendar tool that would allow a coordinator to visualize the relevant information at a glance, making the scheduling decision easier to take.

In terms of technology, it was enforced the use of the Java/Servlet/JSP family of technologies as the foundation for all products. Two computer servers were allocated: the first - for hosting both a file system and a Tomcat Application Server, and the second – for hosting a commercial database server to be accessed by the students' applications. In the dedicated laboratory, a Java environment, a database client and all necessary software were installed on each PC station.

4.2.2. Students' Profile

The subjects of our observational study were the high-level undergraduates students enrolled in the "Software Engineering Studio". The students attending this course had completed the required software engineering courses and had acquired a good level of computing skills.

The class included six teams (five of them being composed of four students and the other one being composed of three students). Each half of the class was assigned to one of the two processes: a process based on XP and UPEDU process. For more accurate results, after evaluating all team portfolios, four teams – each composed of four students – were taken into account for this analysis: two XP teams (X1, X2) and two UPEDU teams (U1, U2).

4.2.3. Data Collections

A portfolio – as a collection of students' collaborative work generated during the Studio – exhibits the students' effort and progress, by containing the entire set of artifacts specific for the development process, as well as the time sheets of the team members. The effort was measured by requiring the students to record the time spent on each activity or artifact produced. An industrial project management tool that included an effort-tracking part was adapted for the students use. The purpose of choosing this tool was to strengthen the effort-gathering process and also to provide students with the feel of using of such a tool.

4.3. Results and discussion

The software development is a set of cognitive activities, which developers perform for the purpose of transforming requirements into actual software systems (Robillard, Kruchten, d'Astous, 2002). This cognitive structure governs the whole of the software development process and implicit – the activity of knowledge acquisition.

Software development is a discipline centered on artifacts. Used to capture specific information on software product, artifacts are considered a direct way of communicating information and an efficient form of knowledge transfer. The information's capture, understanding and communication depend on several factors, including the software development team's cognitive abilities. The unique set of product specifications, the sole time collection tool and, more than that, the alignment of the two processes with focus

on cognitive aspects allows the analysis and the comparison of the students' portfolios contents.

The purpose of this research is to explore the students' knowledge acquisition behavior and to compare the cognitive aspects specific in the case of two development processes. It is supported the claim that "the same type of [cognitive] processes that occur for individuals are conceptually involved in the information processing by the group" (Hinsz, 1990). Therefore, the individual mental model transcends the cognitive facilities of individuals and could be considered a starting point for becoming a team mental model. Going further in this logic, the notion of team mental model was extended at a group level specific to the software process. In this research we extend the concept of mental model to software process and in particular to software artifacts produced in moving through a development cycle, by defining "mental artifact" and "physical artifact" terms.

The UPEDU process introduces students to the software process activities and their corresponding artifacts and enables them to understand the roles played in software development. All performed activities, for capturing the requirements, for presenting the architectural design decisions, or for planning the tests, end in written artifacts. An artifact is a piece of information produced by a role performing an activity during the software development. By using UPEDU process, students apply "cold communication media" to communicate necessary information within and outside the team (Cockburn, 2002).

On the other side, the agile software development philosophy makes the verbal communication one of its main values and de-emphasizes the importance of written documentation. Cockburn contends: "the most effective communication is face-to-face, particularly when enhanced by a shared modeling medium" (Cockburn, 2002). Using XP process, students apply "hot communication media" to communicate necessary information when working together (Cockburn, 2002). Even though, as J. Smith mentions (2001) : "...in the books about the XP approach, the terms "artifact" do not appear in the index.... it's not difficult to read through the text and pick out references that are artifacts". A fundamental message is that artifacts should be produced only when they add the best possible value to the project.

4.3.1. Artifacts effort distribution during the project progress

For allowing a more accurate comparison of students' portfolios, three sets of artifacts were defined. These sets are general enough to fit each process, but on the other side, specific enough to provide a meaningful description of the artifacts composing the set. The three artifacts sets meeting these criteria, as they are presented in Table 4.1, are:

- The set of artifacts specific to the predevelopment stage includes the requested artifacts specific to requirements, to analysis and design, and to technical planning
- The set of artifacts specific to the development stage includes the required artifacts specific to components development and integration, and to testing
- The set of artifacts specific to the managerial stage includes the required artifacts specific to process and project management

Table 4.1 Artifacts mapping table

UPEDU	Artifacts set	XP
Use-Case Model Glossary Design Model Test Plan Test Case	Predevelopment	User Stories Conversation notes CRC Card
Component Implementation Model Test Evaluation Report	Development	Test Code Test Data Unit Code System Code
Iteration Plan Development Plan Measurement Plan Configuration Management Plan	Management	Story Estimates Release Plan Code Management Tool

The effort spent on each artifact set by UPEDU and XP teams as well as the content of all corresponding artifacts was analyzed. To enable comparison, the total effort for each team-project is normalized to 100%. Figure 4.1 shows the normalized effort distribution for the realization of the artifact sets during the project progress, for each of the two processes.

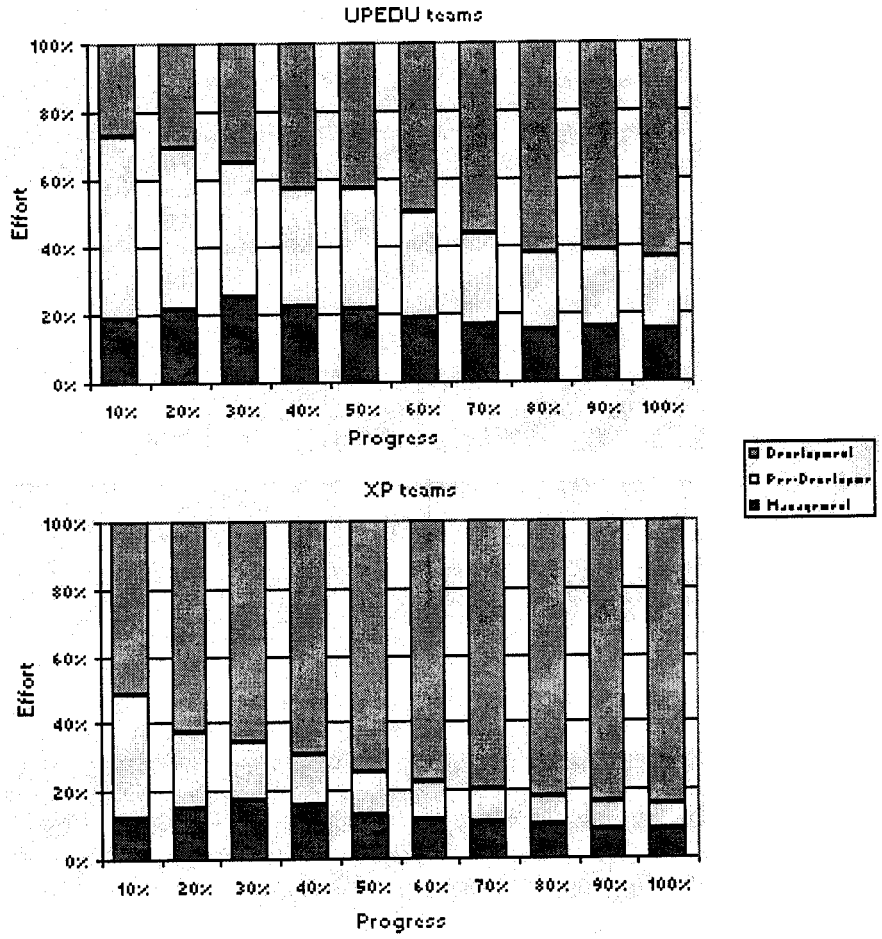

Figure 4.1 Artifacts effort distribution during the project progress

The first remark is that UPEDU teams spent more effort on the artifact set specific to the predevelopment stage than XP teams. A well-defined software process as the UPEDU puts more emphasis on the engineering aspects of the software implementation by stressing the pre-coding activities. There is a systematic approach for the requirements capture, which is materialized in a collection of use-cases, use-case diagrams and a glossary of terms. In the XP process, the features of the system are captured as user stories, broken down lately into small tasks. The analysis of these features is performed

through conversation between the on-site client and the development team. The approach to architecture is quite different between UPEDU and XP. In XP the design model evolves over a series of brainstorming sessions in which developers sketch out the design. In UPEDU process, the use-case realization is the design perspective of the use-case.

The set of artifacts specific to development is related to code construction. It is by far the most effort demanding artifacts set in the case of XP processes. XP strives for simple design and uses a simplifying technique whereby the programmers examine whether the code is reflecting the design in their minds. The use of the "refactoring" technique can increase the effort, as the students have no experience with this technique.

The set of artifacts specific to management follows similar patterns in the case of two processes. The UPEDU process puts more emphasis on managerial activities than XP process, fact that explains the differences in required effort in favor of UPEDU teams.

The Studio setting based on cognitive activities structure allows a more detailed comparison. Since the relative effort on each artifact set ads up to 100%, it is possible to represent those three sets in a plane, by defining a space delimited by three markers, corresponding to the three artifact sets: Predevelopment, Development and Management. The resulting graph is bounded on the three axes by a maximum value of the effort for each artifact set, which is 100% (Germain, 2004). The marker at the top far left is reached when all the effort is expended in producing the management artifacts set, while the marker at the top far right is reached when all the effort is expended in producing the development artifacts set. The third marker at the bottom of the diagram is reached

when all the effort is expended in producing the predevelopment artifacts set. The resulting graph can be seen in the Figure 4.2.

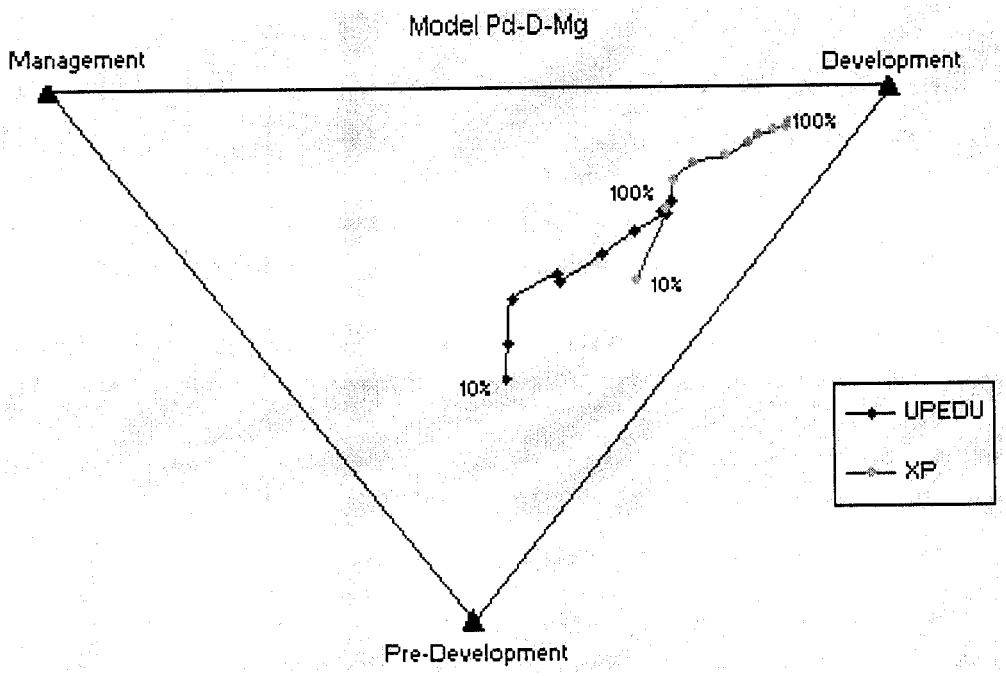

Figure 4.2 Evolution curve in Predevelopment-Development-Management space

The data points curves in the graphic represent the evolution of the relative effort through the software development, for the two processes used, based on the artifacts production effort. The project advancement steps have been defined as the ratio of cumulative effort expended at a given position by the total effort expended in the whole project. It is important to note that due to the cumulative nature of the used data, the effort slip sample is much bigger at the end of the evolution curve. The two progression patterns are coherent, although they differ from one to another, due to the different communication media used among student teammates, in the case of UPEDU group and XP group.

The UPEDU teams have more balanced effort among the three sets of artifacts. For the first tier of the project advancement, the UPEDU teams seem to have their effort invested in all three sets of artifacts. In the very first part of the project the predevelopment artifacts seems to be more influential. An orientation to Development set is observed for the two last tiers of the project advancement. The XP teams have for the first tier of the project advancement their effort more invested in predevelopment and development artifacts set and less in management artifacts set than the UPEDU teams. The programming activity, which starts very early, gives a "head development" style to the XP evolution curve.

4.3.2. Data analysis based on mental and physical artifacts

A thorough look at the distribution of the effort spent on the groups of artifacts underlines a difference between the two processes. When exploring the evolution of the artifacts vs. the effort spent on them by UPEDU teams, some interesting issues and possible explanations emerge. Such an aspect is shown in Figure 4.3. In the case of both predevelopment and development artifact groups UPEDU teams recorded time spent on artifacts, but they did not produce any text document or diagram. Based on the descriptions of the corresponding activities inserted in the time record tool, they spent this time by explaining and understanding each other's mental model. Students shared their own mental model for obtaining a unique model – the team level model.

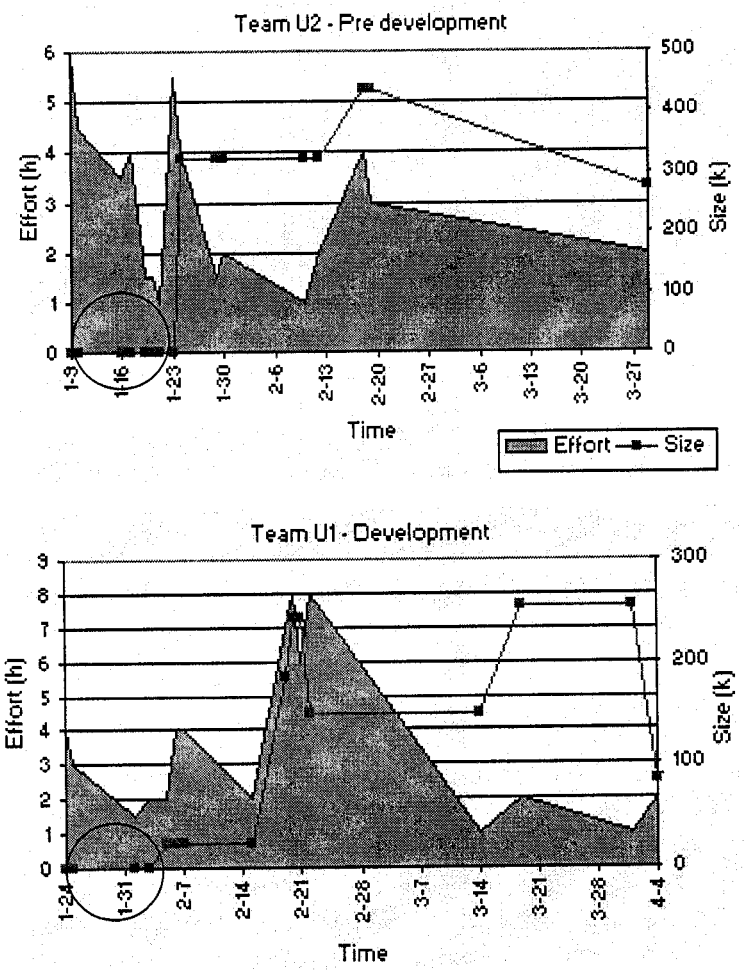

Figure 4.3 Effort vs. artifacts size

The notion of mental model was extended to software process and in particular to software artifacts produced in moving through the development cycle (Dulipovici, Robillard, 2004).

The term "physical artifact" was defined as a piece of information produced by completing an active, creative activity, as shown in Table 4.2. A physical artifact can be a text document, a model – mainly represented by diagrams, or a piece of code. The "physical artifact" represents the explicit knowledge acquired by the students.

Table 4.2 Physical and mental artifact

Activity	Write Draw Code Code & Test Test Integrate & Test Technical Administration	⇒ Physical artifact
Activity	Read Think Discuss Browse/Search Training Inspect/Review	⇒ Mental artifact

The term "mental artifact" was also defined as a piece of information produced by completing a reflexive or an interactive activity, as shown in Table 4.2. A mental artifact is a model that "runs" in the human mind. The "mental artifact" represents the tacit knowledge acquired by the students.

By analyzing each set of artifacts from this point of view, a remark has to be made: the emphasis on mental or physical artifacts shifts from one set to another, all along the project advancement. Figure 4.4 shows the distribution of the effort on mental and

physical artifacts, for the UPEDU and the XP teams, for each of the three sets of artifacts.

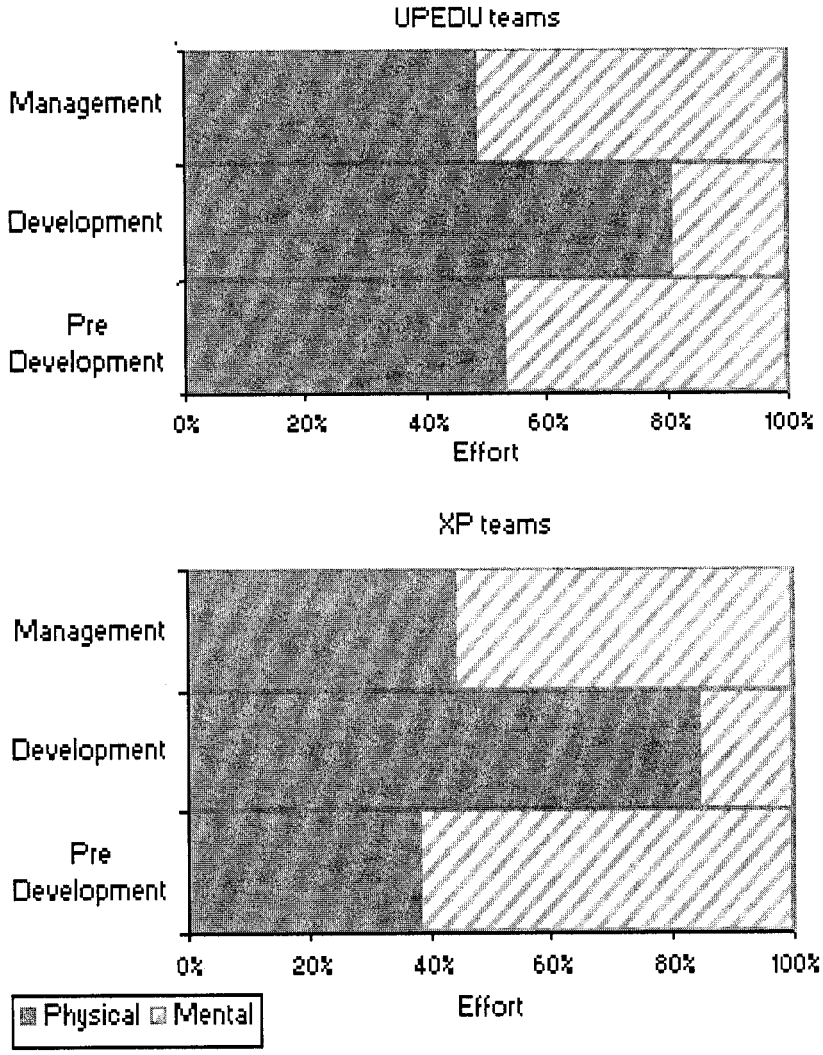

Figure 4.4 Effort distribution on mental and physical artifacts

The biggest difference in the mental vs. the physical pattern is observed in predevelopment artifacts set. XP is known as a back-loaded process, with which the requirements, the analysis, and the design occur by small increments in the construction of the system, in a spontaneous manner. The notion of architecture is replaced by

"metaphor", which explains the essence of how the system works. In UPEDU the architecture is concerned with structure and behavior, being constructed by different views. Mental models are used by students to understand and to explore the problem space and to form a base for communication. Holt (2002) claims "software architecture is most usefully thought of as a mental model shared among the people responsible for software". UPEDU stresses the importance of the drawing up use-case model and the written architecture document. The effort is well balanced between the mental and the physical artifacts.

In the case of development artifacts set both process groups follow the same pattern, in which the physical artifacts dominate the effort for all the teams. A similar pattern is also observed for the management artifact set.

4.3.3. Mental and physical artifacts effort distribution during the project advancement

The analysis continues by referring the effort distribution spent on mental and respectively on physical artifacts over the time, for each of the two processes used. Figure 4.5 shows the normalized effort distribution on mental and physical artifacts all along the project.

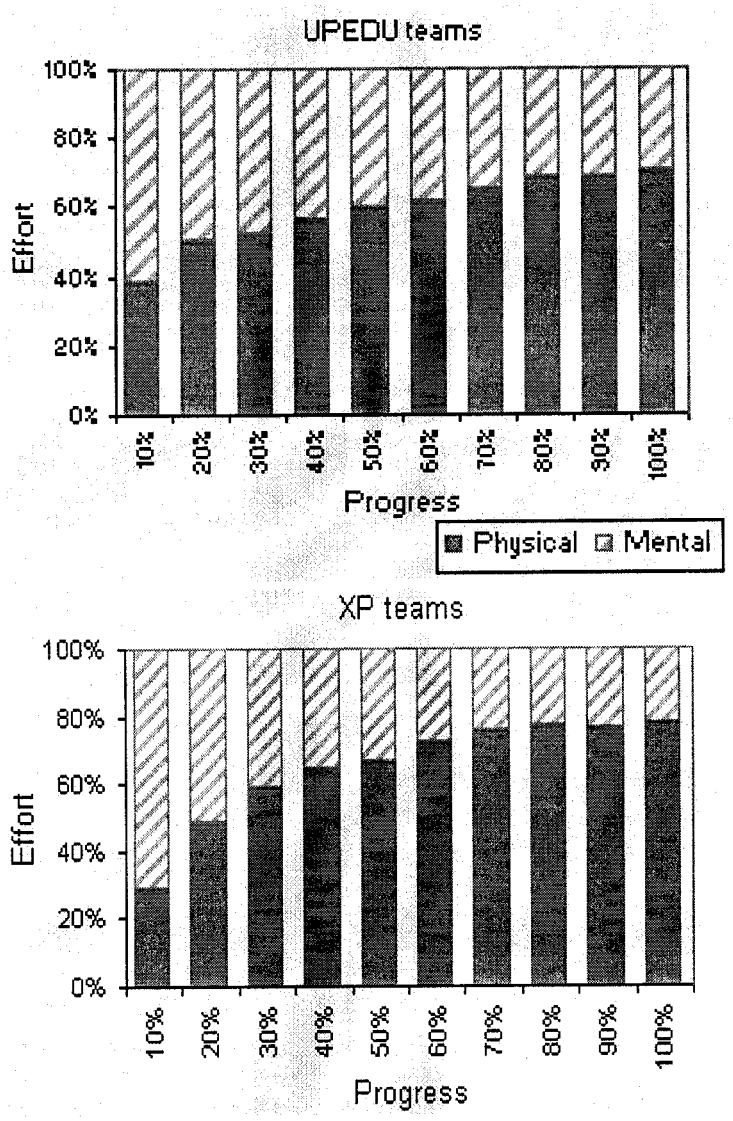

Figure 4.5 Mental and physical artifacts effort repartition vs. project progress

The same profile of mental and physical artifacts repartition can be observed for both processes (UPEDU and XP). Even though XP is based on verbal communication and does not require comprehensive written artifacts, the emphasis is on physical artifacts. A significant difference is observed at the beginning of the project: the XP teams put more mental effort than the UPEDU teams. In this part of the project the emphasis is on

the creation of user stories, and metaphors. All these provide a means for the student teammates to share the individual mental model, to understand it and to create the team mental model.

The Figure 4.6 shows the distribution over the time of the effort spent on mental artifacts specific for each of the three artifacts sets, for the two processes.

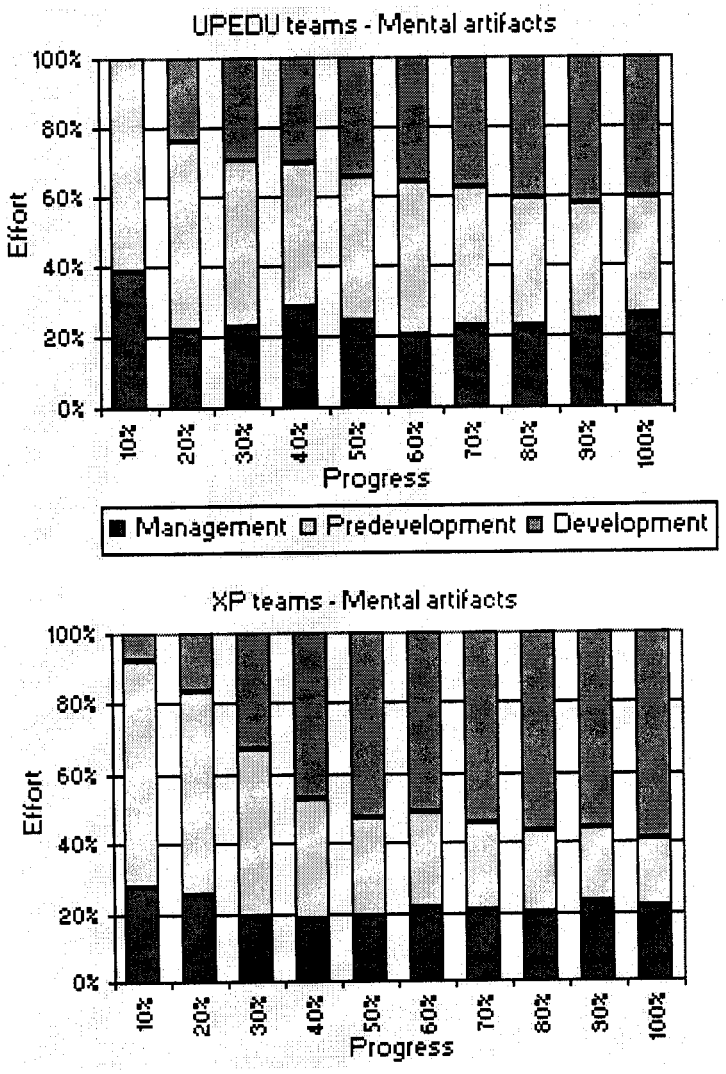

Figure 4.6 Mental artifacts effort distribution during the project progress

Figure 4.7 shows the evolution curve of the effort spent on mental artifacts for the two processes, in the Predevelopment-Development-Management space.

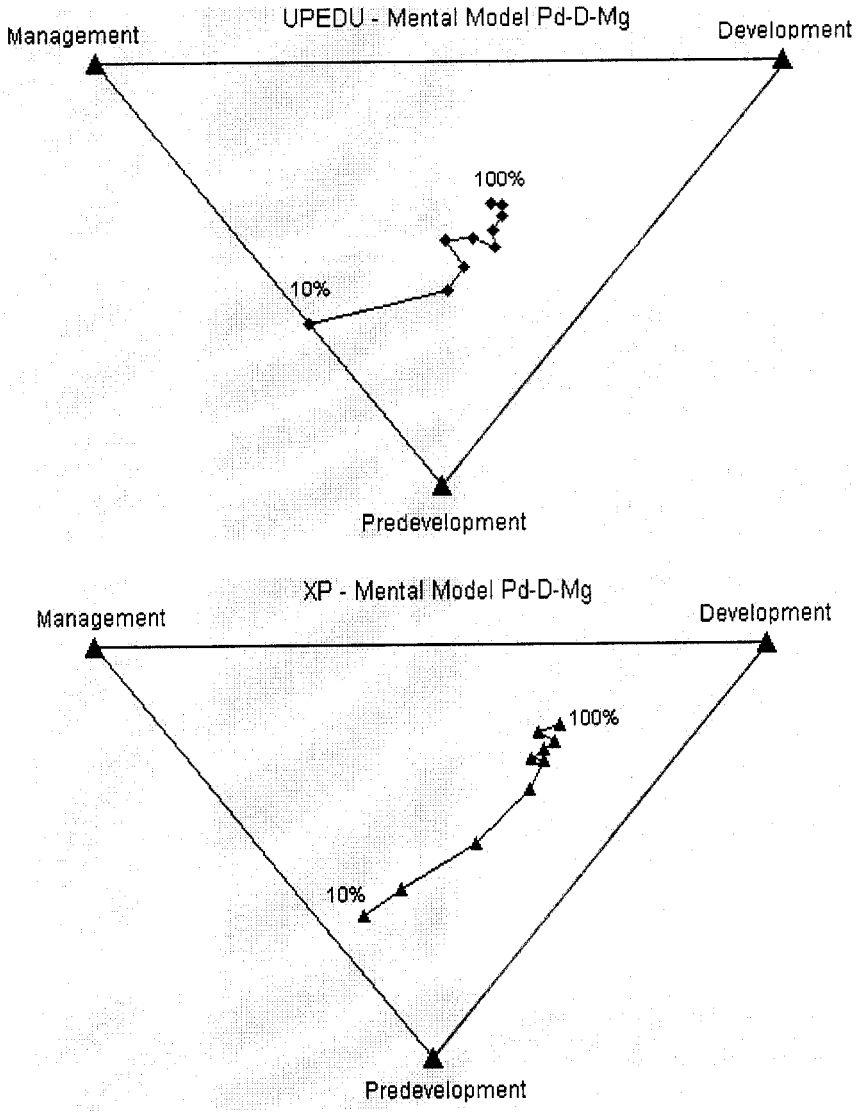

Figure 4.7 Evolution curve for mental artifacts, in the Predevelopment-Development-Management space

At the beginning of the project advancement, the students' effort of the UPEDU teams is invested in mental activity for predevelopment and management artifacts. The

mental artifact for development group starts to be developed later. For the XP teams, the mental artifacts for development start from the beginning of the project and it is the most important effort demanding artifact set for the last part of the project.

The Figure 4.8 shows the distribution over time of the effort spent on physical artifacts specific for each of the three artifact sets, with regard to the two processes.

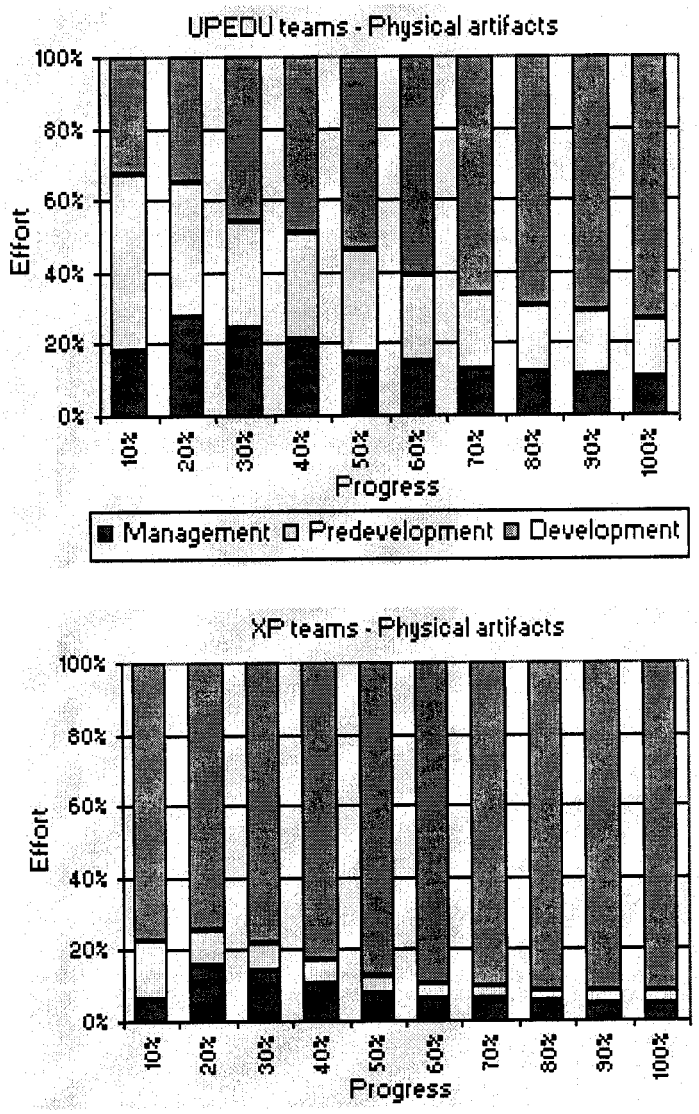

Figure 4.8 Physical artifacts effort distribution during the project progress

Figure 4.9 shows the evolution curve of the effort spent on physical artifacts for the two processes, in the Predevelopment-Development-Management space.

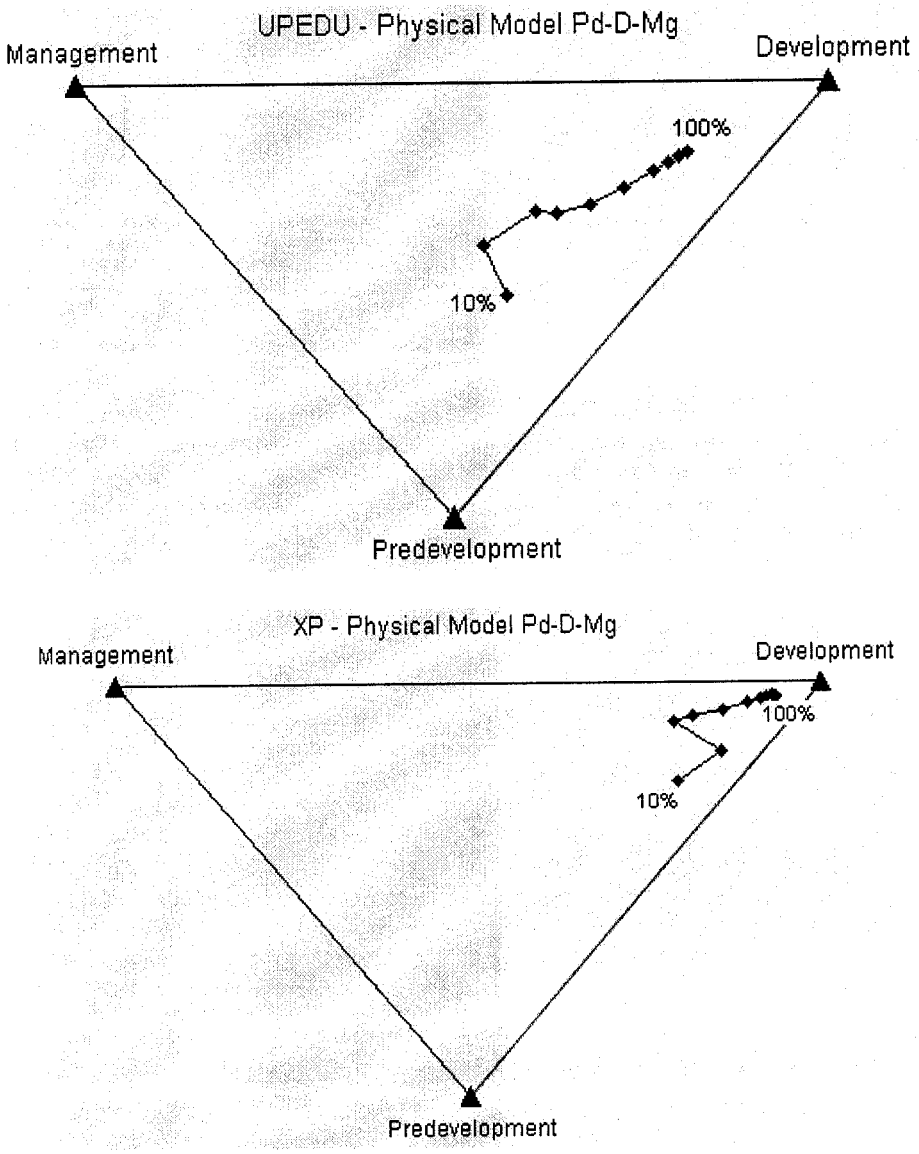

Figure 4.9 Evolution curve for physical artifacts, in the Predevelopment-Development-Management space

The effort on physical artifacts is repeatedly shared out among the three artifact sets, and that for both processes. Due to the big emphases that XP process put on code, the

effort on physical artifacts for the development set is by far the biggest effort all along the project progress. As for the UPEDU process the effort is more balanced for the first tier of the project progress. The artifacts for development gain in importance for the last two tiers of the project progress.

4.4 Conclusion

Software development is more than models and code; it includes knowledge acquisition, understanding and problem-solving. In the particular case of student teams in a project-base course, the students are in the same time learning and solving problems for building a software product by following a defined process. This research allows introspection into the instructional practices for a better understanding of students' cognitive behavior in order to determine the required actions for improving academic projects.

The two software processes used in this observational study recommend the production of different artifacts that form the basis for all communication relating to the software development. The extension of the concept of mental model to software process and the introduction of the terms mental artifact and physical artifact allow the measurement and the analysis of the students' effort on producing the required artifacts.

A first remark is related to the evolution of the artifacts asked to be produced. Students form mental models for exploring problem and solution space. The process used puts more or less emphasis on some mental or physical artifacts depending on the artifact set or on the project progress. XP students shared their mental models by using

"hot communication media". UPEDU recommends the use of "cold communication media" and external graphical representation for the models. UPEDU process is architectural centered; XP is a back-loaded process. Even though, students from both process groups put big effort for developing mental artifacts specific to understanding the requirements, analyzing the alternative solutions and designing the system. Predevelopment mental artifacts are well represented all along the project, with an emphasis in favor of XP group. As for the physical artifacts, XP demands far more effort for producing the development artifact set than the UPEDU process.

For the management artifact set, both process groups follow almost the same pattern during the project progress for producing the mental artifacts and the physical artifacts as well. Different patterns can be observed for the production of artifacts specific for predevelopment set and for development set and that in case of both mental and physical artifacts.

A second remark is related to the specific artifacts asked to be produced by students – each process has its own set of artifacts. According to the data, three categories of artifacts were identified:

- process-specific artifacts used by the same specific process group
- non-process-specific artifacts used by both process groups (UPEDU and XP)
- process-specific artifacts used by the other specific process group

As UPEDU is specifically designed for an academic environment, the UPEDU artifacts are suitable for such an environment. However, some XP practices seem very appropriate for student work. In exchange, XP group used some UPEDU practices and

produced some UPEDU specific artifacts. Students used the practices and produced the artifacts they found useful for their comprehension. The findings of this research suggest that a more balanced process should be used. This process should include in the same frame practices from both processes especially for predevelopment and development stage.

Limitations applicable to this kind of study already mentioned in (Germain, Robillard, 2003) apply here too. First, the validity of the study rests on the level of confidence that exists in the effort slips entered by the students. Some ambiguity, time missing or duplicate data were observed in previous versions of the time slip system. The use of an industrial time collection tool strengthened the data gathering and improved the data quality.

Secondly, the use of two specific processes could limit the generalization of the research results. Nevertheless, the two processes used can be considered representative for the two methodologies – the traditional activity-role-artifact based methodology and the agile methodology.

While recognizing that software development is multifaceted, involving many cognitive factors, the present observational study has not attempted to address all aspects of this complexity. Future work could use different settings to measure students' cognitive behaviors and to test present findings in other pedagogical settings. The research could also be extended by taken into account the students' cognitive style and the students' performance in a process or in another.

Discussing the use of two different process approaches from a cognitive point of view can yield valuable results for the design and the adoption of the most appropriate set of software development practices in an academic environment. The results have implications for educators to prepare students to become effective skilled practitioners and to work as effective members of software development teams.

Acknowledgments

We are grateful to Éric Germain who participated in the preparation of the semester, acted as lecturer for the course and was deeply involved in the cognitive activity classification. We would also like to thank Martin Robillard for his insightful comments while we were building the requirements specification. This project would not have been possible without the participation of the students enrolled in the winter 2002 "Software Engineering Studio" course at École Polytechnique de Montréal.

DISCUSSION GÉNÉRALE ET CONCLUSIONS

La source des idées discutées dans ce mémoire se retrouve dans le grand intérêt des praticiens sur les méthodologies de développement de logiciels (Fayad, 1997; Fugetta, 2000). Cette attention est, en partie, en raison de l'emphase mise sur la qualité des produits logiciels. En outre, la pression du marché pour livrer des logiciels rapidement, en apportant souvent des changements, mais tout en respectant la qualité des produits, force les praticiens à repenser les pratiques traditionnelles de développement de logiciels (Glass, 2001). Deux philosophies principales de développement de logiciel apparaissent. La première philosophie est basée sur le triplet « activité – rôle – artefact ». À cet égard, elle est une approche systématique pour les activités du processus et elle souligne l'importance des artefacts écrits. La deuxième philosophie – dite de type agile – est basée sur une douzaine de pratiques qu'un processus doit respecter.

Il y a beaucoup de différences entre ces deux types de méthodologies, et surtout, concernant la modalité de communiquer l'information tout au long d'un projet logiciel, et ce à l'interne ainsi qu'à l'externe de l'équipe de développement. La méthodologie basée sur le triplet « activité – rôle – artefact » souligne l'importance des artefacts écrits pour communiquer l'information. Contrairement à cette méthodologie, les méthodologies de type agile et notamment XP, font de la communication verbale une valeur principale et propose une approche minimaliste à la documentation écrite. De plus, le choix d'un processus s'avère très difficile à cause du grand nombre de facteurs que l'on doit considérer.

D'ailleurs, le développement d'un système logiciel exige une compréhension appropriée non seulement des aspects techniques reliés au processus de développement, mais aussi du facteur humain impliqué dans ce processus. Stroustrup – l'auteur du langage de programmation C++ – remarque que « design and programming are human activities; forget that and all is lost ». (mentionné par Booch et al., 1998, p.93). En effet, le processus développement logiciel, dans son ensemble, a un caractère fortement cognitif. Comprendre les aspects cognitifs qui sont entrelacés dans les activités du processus s'avère d'une importance majeure. Nous pourrions même soutenir que les structures cognitives des développeurs dirigent tout le processus de développement.

Les résultats de notre étude empirique observatoire démontrent les particularités des aspects cognitifs propres à la création des artefacts spécifiques à deux processus : UPEDU (Unified Process for Education) et XP (Extreme Programming). Bien que les deux processus utilisés soient très différents de point de vue de la communication de l'information, nos résultats qualitatifs montrent que le même effort est dépensé sur les artefacts mentaux ou sur les artefacts physiques, et ce dans le contexte des deux processus. Le processus utilisé met plus ou moins l'accent sur certains artefacts mentaux ou physiques, selon le groupe d'artefacts ou selon l'évolution des artefacts tout au long du projet logiciel. Un modèle cognitif à trois pôles nous a permis de rédiger la courbe d'évolution de l'effort demandé par la création des artefacts. La différence entre les courbes pour les deux processus est en raison de la modalité différente de communiquer l'information. Par ailleurs, tant les équipes XP que les équipes UPEDU ont démontré un comportement qui soutient l'existence d'un modèle mentale d'équipe. Une différence

considérable entre les deux courbes d'évolution est observée vers la fin du projet et elle fait référence à la partie d'évolution du produit.

Notre étude empirique observatoire a également démontré un lien possible entre le type de processus de développement et la qualité de réalisation des requis. La méthodologie de développement basée sur le triplet « activité – rôle – artefact » semble plus appropriée pour développer les entités d'application basées sur des requis à caractère dynamique. D'autre part, la méthodologie de développement agile semble plus appropriée pour développer les entités d'application basées sur des requis à caractère statique.

Un des apports principaux de ce travail de recherche est à la fois l'originalité de l'approche de recherche et les résultats obtenus. Notre approche centrée sur les activités cognitives et les artefacts mentaux et physiques permet la comparaison des processus. Quant aux résultats obtenus, ils permettent une meilleure compréhension du rôle joué par un processus spécifique et ils prouvent une fois de plus l'importance des aspects cognitifs en processus logiciel. La connaissance des processus mentaux propres à une activité spécifique aide à concevoir des règles d'art appropriées et efficaces pour cette activité. Toutefois, il n'existe pas de démarche universelle applicable à toutes les situations mais quelques approches adaptées seraient un gain significatif (Carver, 2003).

Nos recherches présentent toutefois certaines limites qui sont inhérentes à toute étude empirique. Tel que souligné par Robillard (1996) « It is utopic to believe that an industrial environment can be simulated in a classroom. ... The project and the process should be down-sized and adapted to the proper environment. ». Les études en génie

logiciel conduites dans un environnement universitaire et utilisant des étudiants comme sujets, apportent leurs contributions de valeur au génie logiciel empirique. Tandis que les sujets abordés dans cette recherche ont peu d'évidence empirique, nous considérons que les résultats présentés sont davantage importants aux praticiens et ouvrent la porte à d'autres études comparatives. Cette recherche est un point de départ et seule la recherche future fondée sur les résultats de ce travail permettra de les fructifier.

Plusieurs avenues de recherche futures pourraient être identifiées. Une première avenue consiste à raffiner les caractéristiques et les classifications des activités cognitives et des artefacts utilisés dans notre recherche ainsi que de renforcer l'approche de saisie de l'effort. À cet égard, un nouvel outil a été conçu pour être utilisé dans le cadre de l'Atelier Génie Logiciel. La liste proposée pour les artefacts à produire, ainsi que des critères plus stricts de validation de jetons d'effort permettent une plus grande objectivité concernant la validation des données. Une deuxième avenue consiste à répéter l'étude dans un contexte industriel ainsi que sur plusieurs projets.

Pour conclure, la controverse sur l'existence de la balle d'argent se poursuit dans la communauté des développeurs logiciels (Beck et Boehm, 2003). Cependant, nous croyons que, si cette balle existe, elle se trouve plutôt dans la compréhension des aspects cognitifs du travail des développeurs. Dans ce sens, Boehm affirme :

> « Methods are important, but potential silver bullets are more likely to be found in areas dealing with people, values, communication » (2003, p.24)

BIBLIOGRAPHIE

AGILE ALLIANCE. 2004. « Agile alliance web site ». [En ligne] http://www.agilealliance.org (Page consultée le 18 mars 2005).

ARMOUR, Philip G., 2003, « The Laws of Software Process: A New Model for the Production and Management of Software », Auerbach Publications

BASADUR, M.S., and GELADE, G. ,2002, « Knowing and Thinking: A New Theory of Creativity », Management of Innovation and New Technology Research Centre Working Paper No.105, McMaster University, Hamilton, On. Canada

BASILI, V.R, 1990, « Viewing maintenance as reuse-oriented software development », IEEE Software, Volume:7:1, p.19 - 25

BECK, Kent. 1999. « Embracing change with Extreme Programming ». IEEE Computer. 32:10. p. 70-77.

BECK, Kent. 2000. « Extreme Programming Explained : Embrace Change ». Boston: Addison Wesley. 190 p.

BECK, Kent, BOEHM, Barry. 2003 « Agility through discipline: A debate ». IEEE Computer. 36: 6. p.44-46.

BLISS J., OGBORN J., 1989, « Tools for exploratory learning: A research programme » Journal of Computer Assisted Learning v.5:1, p.37–50

BOEHM Berry, TURNER Richard, 2003, « Balancing Agility and Discipline: A Guide for the Perplexed », Pierson Education

BOOCH G., MARTIN R., NEWKIRK J., 1998, « OO Analysis and Design with Applications », Addison Wesley Longman

BROOKS, Jr., Frederick P. 1987. « No silver bullet : Essence and accidents of software engineering ». IEEE Computer. 20: 4. p.10-19.

CARVER Jeff, SHULL Forest, BASILI Victor, 2003, « Observational studies to accelerate process experience in classroom studies: an evaluation », Proceedings of the International Symposium on Empirical Software Engineering (ISESE'03)

CHANDRASEKARAN B., KAINDL H., 1996. « Representing functional requirements and user-system interactions », AAAI-96 Workshop on Modeling and Reasoning about Function, Portland, Oregon

CHARETTE R., « The decision is in: Agile versus Heavy Methodologies », 2001, Cutter Consortium, e-Project management Advisory Service, v2:19, p1-3

CHEN, Z., 1992, « Human aspects in object-oriented design: an assessment and a methodology », Behaviour & Information Technology, vol. 11(5), pp. 256-261

COCKBURN, Alistair, WILLIAMS, Laurie, 2001. « The costs and benefits of pair programming ». Extreme Programming Examined. Sous la direction de Giancarlo Succi et Michele Marchesi. Boston : Addison Wesley. P. 223-247.

COCKBURN, Alistair. 2002, « Agile Software Development ». Boston, Mass. Addison Wesley. 278 p.

COCKBURN, Alistair, 2003, « The Cooperative Game manifesto for software development », Alistair Cockburn web site ». [En ligne]

http://alistair.cockburn.us/crystal/articles/ (Page consultée le 18 mars 2005).

COCKBURN, Alistair, 2004, « The End of Software Engineering and the Start of Economic-Cooperative Gaming », Computer Science and Information Systems, v1:1, p.1-32

COVINGTON Michael, 1985, « Documentation that works », PC Technical Journal, vol. 3, no.1

CUGOLA, Gianpaolo, GHEZZI, Carlo. 1998. « Software processes : A retrospective and a path to the future ». Software Process - Improvement and Practice. New York : Wiley InterScience. P. 101-123.

DOYLE J., FORD D., 1999, « Mental Model Concepts Revisited: Some Clarifications and a Reply to Lane », System Dynamics Review. 15(4), p. 411-415.

DULIPOVICI Mihaela, ROBILLARD Pierre N., 2004a, « Cognitive aspects in a project-based course in software engineering », Proceedings of 5th ITHET Conference, Istambul, Turkey, p.353-359

DULIPOVICI Mihaela, ROBILLARD Pierre N., 2004b, « L'influence du processus sur la qualité du produit », Journal d'Ingénierie des Systèmes d'Information, numéro spécial « Information Systems Quality », vol. 9 / 5-6, p. 103-116

ECOLE POLYTECHNIQUE DE MONTRÉAL. 2002. « UPEDU ». [En ligne].

http://www.yoopeedoo.org (Page consultée le 18 mars 2005).

FAYAD M., 1997, « Software development process: a necessary evil ». Communications of the ACM, v. 40:9, p.101-103

FUGETTA, Alfonso. 2000. « Software process : A roadmap ». The Future of Software Engineering 2000 : 22nd International Conference on Software Engineering. Sous la direction de Anthony Finkelstein. NY : Association for Computing Machinery. p. 25-34.

GAGNÉ, Robert. M., 1985, « The Conditions of Learning and Theory of Instruction », Holt, Rinehart & Winston Ed., New York

GARVIN, D., 1984, « What does "product quality" really mean? », Sloan Management Review vol. 26(1), p. 25–45

GERMAIN, Éric, DULIPOVICI, Mihaela, ROBILLARD, Pierre N. 2002a. « Measuring software process activities in student settings ». Proceedings of the 2nd ASERC Workshop on Quantitative and Soft Computing Based Software Engineering. (QSSE'02) Edmonton : Alberta Software Engineering Research Consortium. p. 44-49

GERMAIN, Eric, ROBILLARD, Pierre N., DULIPOVICI, Mihaela, 2002b. « Process activities in a project based course in software engineering ». 32nd ASEE/IEEE Frontiers in Education Conference. Piscataway : IEEE, p. S3G-7-S3G-12.

GERMAIN, Eric, ROBILLARD, Pierre N. 2003. « What cognitive activities are performed in student projects ». Proceedings of the 16th Conference on Software Engineering Education and Training (CSEET'03). Los Alamitos : IEEE Computer Society. p. 224-231.

GERMAIN, Eric, ROBILLARD, Pierre N. 2004a « Engineering-based processes and agile methodologies for software development : A comparative case study ». The Journal of Systems and Software. Acceptée pour publication.

GERMAIN, Eric, 2004b, « Analyse comparative de la répartition de l'effort dans le cadre de processus logiciel », École Polytechnique de Montréal, mémoire de maîtrise

GLASS, Robert L. 2001. « Agile versus traditional : Make love, not war ! » Cutter IT Journal. 14: 12. p. 12-18.

HACKER W. 1994, « Action regulation theory and occupational psychology », The German Journal of Psychology, v. 18:2, p.91-120

HALLING, Michael, ZUSER, Wolfgang, KÖHLE, Monika, BIFFL, Stefan. 2002. « Teaching the Unifed Process to undergraduate students ». Proceedings of the 15th Conference on Software Engineering Education and Training (CSEET'02). IEEE Computer Society. p. 148-159.

HEDIN Gorel, BENDIX Lars, MAGNUSSON Boris, 2003, « Teaching extreme programming to large groups of students », The Journal of Systems and Software, article in press

HIGHSMITH, James A. 2000. « Adaptive Software Development : A Collaborative Approach to Managing Complex Systems ». New York : Dorset House Pub. 358 p.

HIGHSMITH, James A. 2002. « Agile Software Development Ecosystems ». Boston : Addison Wesley. 404 p.

HILL Robert, LEVENHAGEN Michael, 1995, « Metaphors and mental models: sensemaking and sensegiving in innovative activities », Journal of Management, v2, n.6

HINSZ Verlin B., 1990, « Cognitive and consensus processes in group recognition memory performance », Journal of Personality and Social Psychology

HOLT R., 2002, « Software Architecture as a shared mental model », International on Program Workshop Comprehension, Paris, France,

HUDSON Williams, 2003, « Mental Models, Metaphors and Design », [En ligne]. http://www.syntagm.co.uk/design/articles.htm (Page consultée le 18 mars 2005)

JEFFRIES, Ron, ANDERSON, Ann, HENDRICKSON, Chet. 2001. « Extreme Programming Installed ». Boston : Addison Wesley. 265 p.

JOHNSON-LAIRD, P., « Mental model », 1983, Cambridge, Massachusetts, Harvard University Press,

KITCHENHAM B., PFLEEGER S. L., 1996, « Software Quality: The Elusive Target », IEEE Software, vol. 13(1), p. 12-21

KLIMOSKI Richard, MOHAMMED Susan, 1994, « Team mental model: construct or metaphor? », Journal of Management, v.20, n.2

KRUCHTEN, Philippe. 2000. « The Rational Unified Process : An Introduction. » 2nd Ed. Reading : Addison Wesley. 298 p.

LEVESQUE L, WILSON J M, WHOLEY D R, « Cognitive divergence and shared mental models in software development project teams », Journal of Organizational Behavior, v.22, issue 1, Special Issue: Shared Cognition . p. 135-144, 2001

LINDVALL, Mikael, RUS Ioana, 2000, « Process Diversity in Software Development », IEEE Software, July/August, p. 14-18

MATHIANSSEN Lars, PURAO Sandeep, 2002, « Educating reflective system developers », Journal of Information System, 12, p.81-102

MOHAMMED S, DUMVILLE B C, « Team mental models in a team knowledge framework: expanding theory and measurement across disciplinary boundaries », Journal of Organizational Behavior, v.22, issue 2, p. 89-106, 2001

NAUR P., 1992, « Computing: A Human Activity », Reading Massachusetts, Addison Wesley

NEWMAN Williams, LAMMING Michael G., 2003, « Interactive System Design », Addison-Wesley Longman Publishing Co, Boston, MA, 468 pages

NONAKA Ikujiro., TAKEUCHI H., 1995, « The Knowledge-Creating Company », Oxford University Press

NONAKA Ikujiro, TOYAMA Ryoko, KONNO Noboru, 2000, « SECI, Ba and Leadership: a Unified Model of Dynamic Knowledge Creation », Long Range Planning v. 33, p. 5-34

NORMAN Donald A. 1983, « Some observations on mental models »,. In Dedre Gentner and Albert L. Stevens Editors., Mental Models,. Lawrence Erlbaum Associates, Hillsdale, NJ,. P.7–14

NORMAN, Donald. A., 1986, « Cognitive Engineering », Norman & Draper Editors

OSTERWEIL, Leon J. 1987. « Software processes are software too ». Proceedings, 9th International Conference on Software Engineering. New York : IEEE Computer Society Press. p. 2-13.

PALMER, Stephen R., FELSING, John M. 2002. « A Practical Guide to Feature-Driven Development ». Upper Saddle River : Prentice Hall PTR. 271 p.

PAYNE Stephen J.. 1991, « A descriptive study of mental models »., Behaviour & InformationTechnology, 10(1): p.3–21.

PAYNE Stephen J.. 1992, « On mental models and cognitive artifacts ». In Rogers et al., p.103–118.

RAUTERBERG Mathias, 1999. « Activity and Perception: an Action Theoretical Approach », Proceedings of the International Conference "Problems of Action and Observation", submitted to : Cybernetics & Human Knowledge

RENK J., BRANCH R., CHANG E., 1993. « Visual information strategies in mental model development » Visual literacy in the digital age: Selected readings of the 25th annual conference of the International Visual Literacy Association, Braden&Clark-Baca Ed., NY, p.81–91

RISING, Linda, JANOFF, Norman S. 2000. « The Scrum software development process for small teams ». IEEE Software. 17: 4. p. 26-32.

ROBILLARD, Pierre N. 1996. « Teaching software engineering through a project-oriented course ». Proceedings of the 9th Conference on Software Engineering Education (CSEE'96). IEEE Computer Society. p. 85-94.

ROBILLARD, Pierre N., 1998. « Measuring team activities in a process-oriented software engineering course ». Proceedings of the 11th Conference on Software Engineering Education and Training (CSEET'98). IEEE Computer Society. p. 90-101.

ROBILLARD Pierre N., 1999, « The Role of Knowledge in Software », Communications of the ACM, Vol 42, No 1, pp 87-92.

ROBILLARD, Pierre N., KRUCHTEN, Philippe, d'ASTOUS, Patrick., 2001, « YOOPEEDOO (UPEDU) : A process for teaching software process ». Proceedings of the 14th Conference on Software Engineering Education and Training (CSEET'01). IEEE Computer Society. p. 18-26.

ROBILLARD, Pierre N., KRUCHTEN, Philippe, D'ASTOUS, Patrick. 2002. « Software engineering process with the UP/Edu » Boston : Addison Wesley. 346 p.

SAMETINGER J., SCHIFFER S., 1995, « Design and Implementation Aspects of an Experimental C++ Programming Environment », Software Practice and Experience, v.25, no.2, John Wiley & Sons

SASSE Martina-Angela. 1992, « Users' models of computer systems ». In Rogers et al. London Academic Press, p. 225–239

SCHNEIDER Jean-Guy, JOHNSTON Lorraine, 2004, « eXtreme Programming – helpful or harmful in educating undergraduates? », The Journal of Systems and Software, article in press

SMITH J., 2001 « A Comparison of RUP and XP », Rational Edge.

STACY Webb, MACMILLAN Jean, 1995, « Cognitive Bias in Software Engineering », Communications of the ACM, v.38:6, p.57-63

STAGERS, N., NORCIO, A.F. 1993, « Mental models: concepts for human-computer interaction research », International Journal of Man - Machine Studies, 38, p.586 -605

STANDISH GROUP. 2001. « Chaos chronicles version 2.0 ». 316 p.

STAPLETON, Jennifer. 1997. « DSDM, Dynamic Systems Development Method : The Method in Practice ». Harlow : Addison Wesley. 192 p.

STOREY, M.-A.; FRACCHIA F.D.; MULLER H.A. 1999, « Cognitive design elements to support the construction of a mental model during software visualization », Journal of Software Systems, Vol. 44, p.171-185.

USREY, M., and DOOLEY, K. 1996, « The Dimensions of Software Quality », Quality Management Journal, vol. 3(3), p. 67-86

VYGOTSKY L. 1978, « Mind in Society », Harvard University Press, Cambridge, Ma.

VON GLASERSFELD, Ernst, « A constructivist approach to teaching », 1995, In Leslie P. Stete and Jerry Gale, editors, Constructivism in Education, Lawrence Erlbaum Associates, Hillsdale, NJ. P. 3–16

WELLS, Don., 2003, « Don't solve a problem before you get to it ». IEEE Software, 20: 3. p. 44-47.

WILLIAMS, Laurie, KESSLER, Robert R., CUNNINGHAM, Ward, JEFFRIES, Ron. 2000. « Strengthening the case for pair programming ». IEEE Software. 17: 4. p. 19-25.

WILLIAMS, Laurie, UPCHURCH, Richard L. 2001. « In support of student pair-programming ». Technical Symposium on Computer Science Education, Proceedings of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education. New York : ACM Press. P. 327-331.

WOHLIN Claes, RUNESON Per, HÖST Martin, Ohlsson Magnus, REGNEL Bjöorn, WESSLEN Anders, « Experimentation in software engineering: an introduction », 2000, Kluwer Academic Publishers, Norwell, MA, USA

YEN J, FAN X, WANG S R, CHEN C, KAMALI K, 2003, « Implementing Shared Mental Models for Collaborative Teamwork », Working Notes of the Workshop on

Collaboration Agents: Autonomous Agents for Collaborative Environments at IEEE/WIC International Conference on Intelligent Agent Technology (IAT2003), Halifax, Canada.

YOUNG, R. M., 1983, « Surrogates and Mappings: two Kinds of Conceptual Models for Interactive Devices », Gentner & Stevens Eds, Mental models, Hillsdale, New Jersey

ANNEXE

RÉSUMÉS DES ARTICLES DE CONFÉRENCE

MEASURING SOFTWARE PROCESS ACTIVITIES IN STUDENT SETTINGS

Éric Germain, Mihaela Dulipovici, Pierre N. Robillard

2nd ASERC Workshop on Quantitative and Soft Computing Based Software Engineering, QSSE 2002, February 2002, Edmonton, Ab., Canada

Abstract

The recent emergence of various software processes facilitates process activity measurement and provides an opportunity for improving project planning as well as estimating related activities. As such, process measurement of effort spent during each activity is required to better understand the relationship between the various activities. The measurement of activity effort can lead to better understanding of the roles of these activities within the process discipline. Measurement of activity effort can also help to validate the prescribed activities and enable some modifications of the process disciplines.

However, direct measurement of activities performed can lead to measurement interference with the experiment. Predefinition of activities to be used by participants for data recording can only be made under the hypothesis that they will restrict their actions to the prescribed activities. Also, use of activity as a basis for measurement

leads to process data that cannot be compared directly between different processes. Therefore, a classification is needed that will allow easy data entry and map naturally to many different process activity sets while avoiding interference.

This paper describes the process measurement that takes place every winter semester in the Software Engineering Studio class at École Polytechnique de Montréal. Each project lasts 675 hrs-pers. and is completed in parallel by teams of 5 students. Each of the three to five teams, depending on enrolment, base their work on the same formal specification and use a well-defined software process.

The paper presents the activity classification that was used in the 2001 setup as well as the one to be used for 2002. In the first case, the software process used was a variation of the Unified Process for Education (UPEDU), derived from the Rational Unified Process (RUP) in collaboration with Rational Software. In the second, approximately half of the teams will use a coarser variation of the UPEDU, while the other teams will use a process adapted from the extreme programming (XP) methodology. Challenges and difficulties encountered are presented with a brief overview of the findings from the measurements taken in the first setup.

PROCESS ACTIVITIES IN A PROJECT BASED COURSE IN SOFTWARE ENGINEERING

Éric Germain, Pierre N. Robillard, Mihaela Dulipovici

32nd ASEE/IEEE Frontiers in Education Conference, FIE 2002, November 2002, Boston, Ma., US

Abstract

"Studio in Software Engineering" is a curriculum component for the undergraduate-level software engineering program at École Polytechnique de Montréal. The main teaching objective is to develop in students a professional attitude towards producing high quality software. The course is based on a project approach in a collaborative learning environment. The software development process used is based on the Unified Process for EDUcation, which is customized from the Rational Unified Process. An insight into the dynamics of three teams involved in the development of the same project allows us to present and interpret data concerning the effort spent by students during particular process activities. The contribution of this paper is to illustrate an approach involving qualitative analysis of the effort spent by the students on each software process activity. Such an approach may allow the development of a model that would lead to effort prediction within a software process in order to designate the actions for improving academic projects.

DESIGNING AND VALIDATING EMPIRICAL SOFTWARE PROCESS STUDIES IN STUDENT PROJECT SETTINGS

Éric Germain, Mihaela Dulipovici, Sébastien Cherry, Pierre N. Robillard

CUSEC 2003, Janvier 2003, Montréal, Canada

Abstract

This paper presents a series of empirical studies that have or will be undertaken in the context of an undergraduate project course that takes place every winter session at École Polytechnique de Montréal, within the Software Engineering Program. The course, called "Software Engineering Studio", involves the development of a genuine implementation of a specification supplied at the beginning of the semester by several teams of four to six students each. The development work is based on the utilization of the UPEDU, a software process derived from the Rational Unified Process for the purpose of teaching software processes to software engineering students. The paper focuses on the difficulties that often make harder the process of extracting valid results from such studies and on the best practices to minimize the impact of the difficulties encountered. Findings on the understanding of the process by the students and on the relevance of the process will also be discussed. Empirical studies of software process in student settings can bring unique advice on the effective behaviour of software development professionals-to-be.

COGNITIVE ASPECTS IN A PROJECT-BASED COURSE IN SOFTWARE ENGINEERING

Mihaela Dulipovici, Pierre N. Robillard

5th International Conference on Information Technology Based Higher Education and Training, ITHET'04, June 2004, Istanbul, Turkey

Abstract

This paper presents an original approach for identifying and analyzing cognitive aspects specific to artifacts creation in the context of a senior-level project-course. We analyzed the effort spent by student teams on each specific artifact meant for two different processes carried on in parallel: UPEDU (Unified Process for Education) – the traditional activity-role-artifact based methodology and XP (Extreme Programming) – an agile methodology. For this purpose we extend the notion of mental model to software process and we propose to consider the terms of mental artifact and physical artifact. Even though the two used processes are very different, our results show that the same effort is spent on mental artifacts as on physical artifacts, in the context of both processes. The comparison results may allow a better understanding of students' cognitive behavior, in order to designate the required actions for improving academic projects.